AD 684 831

COMPUTER SYSTEMS FOR TEACHING COMPLEX CONCEPTS

Wallace Feurzeig

Bolt Beranek and Newman, Incorporated
Cambridge, Massachusetts

March 1969

BOLT BERANEK AND NEWMAN INC

CONSULTING · DEVELOPMENT · RESEARCH

COMPUTER SYSTEMS FOR TEACHING COMPLEX CONCEPTS

FINAL REPORT

Wallace Feurzeig

March 1969

Department of the Navy
Office of Naval Research
Washington, D. C. 20360

Contract Nonr-4340(00)

FINAL REPORT

COMPUTER SYSTEMS FOR TEACHING COMPLEX CONCEPTS

BBN Report No. 1742

Wallace Feurzeig
Bolt Beranek and Newman Inc.
50 Moulton Street
Cambridge, Massachusetts 02138

March 1969

Submitted to:

Department of the Navy
Office of Naval Research
Washington, D. C. 20360

Contract Nonr-4340(00)

TABLE OF CONTENTS

1. Introduction

The research reported here concerns the use of computers in teaching and learning problem-solving concepts and skills. The approaches taken are multiple because the process of solving problems and the difficulties encountered by students are specific to the subject studied; different ways of using computers are appropriate to teaching problem-solving in mathematics, in physics, in medicine, and in navigation, and we are interested in each of these subjects.

Computers can be programmed to serve students in radically different ways. The computer can serve the student as a master or as a slave. In programmed teaching, the computer controls the interaction with the student. The expression "computer-assisted instruction" is often used to describe applications of this sort in which the computer is used as a drill instructor, tester, or specialized tutor. There are applications of the opposite kind, where the student controls the machine. One obvious example is the teaching of computer programming itself. Another is the use of a computer to simulate a physical laboratory.

There is a potentially rich spectrum of intermediate arrangements in which the computer and student share in directing and controlling the interaction. One family of examples is in the use of a computer to monitor a student's work as he uses a programming language to solve a problem in his own way. The instructional monitor attempts to detect, and possibly to diagnose and remedy, the student's difficulties.

Four instructional programming systems, illustrating these different lines of approach, were designed and developed during the course of the research. The body of this report describes these systems and illustrates their application. The systems are called Mentor, Stringcomp, Simon, and Logo.

Mentor was designed for use in programmed teaching of complex investigations such as detailed case studies. Examples of its use with problems in clinical medicine, business, educational administration, and criminal investigation are included. *Stringcomp* was designed to extend and enhance programmed teaching and testing and also for use as an instructional slave, particularly in formal problem areas such as mathematics, science, and various technical subjects. *Simon* is an instructional monitor. *Logo* is a simple programming language expressly designed for teaching mathematical thinking.

The research was carried out over a five-year period under the technical direction of Wallace Feurzeig and John A. Swets. Many Bolt Beranek and Newman staff members and consultants collaborated in performing the work. The following list credits the major individual contributions.

The Mentor system was designed by Feurzeig and Daniel G. Bobrow. Programming of the first version of Mentor was performed by Myra Breen; subsequent system programming was done by Cynthia Solomon and Frank Frazier. John A. Swets and Richard H. Bolt contributed ideas that were incorporated in the design and in applications. Applications of Mentor were programmed by Feurzeig, Breen, and Solomon. The medical problem was designed by Dr. Preston Munter of the Harvard University Health Center; the educational

administration problem by Robert Ruderman of Educational Testing Services, the business problem by students of the Harvard Business School, the mystery problem by Feurzeig and Robert Donaghey.

The Stringcomp system was designed by Feurzeig, Bobrow, and Professor Seymour A. Papert of the Massachusetts Institute of Technology, a consultant. The system was programmed by John Barnaby and Paul Wexelblat. Feurzeig and Philip Faflick designed the applications and they were programmed by Faflick.

The Simon system was designed by Feurzeig, Wexelblat, and Professor Ronald C. Rosenberg of the Massachusetts Institute of Technology, a consultant. Applications were designed by Rosenberg and programmed by Wexelblat and Faflick.

The Logo system was designed by Feurzeig, Papert, and Bobrow. Major contributions were made by Solomon and by Richard J. Grant. The system was first programmed by Bobrow in a preliminary version. Subsequent programming was performed by Charles R. Morgan and Grant. Classroom teaching applications of Logo were made by Papert and Solomon.

This report was organized and put together by Feurzeig. Frazier, Papert, and Rosenberg made major contributions to the writing. Pearl Stockwell served as editor and did the technical typing. We wish to extend our appreciation to Dr. Glenn L. Bryan and to Dr. Victor Fields, the Office of Naval Research scientific officers of this project, for their advice and counsel throughout the effort.

2. The Mentor Language for Programmed Discourse

One sort of ideal for workers in the field of computer-assisted instruction (CAI) is intelligent artificial teaching. Contemporary work in teaching with computers has had more limited and modest educational goals. Most work has teaching objectives such as drill and practice, testing factual information, or tutoring in the sense of classical branching programmed instruction. Thus, there has been a large gap between some workers' fond expectations and their current practice, and pundits, seizing their opportunity, have parodied current work along lines such as this:

Computer: Roses are red. Violets are blue.
What color are roses?
(a) Red
(b) Blue
(c) Violet
(d) None of the above

Student: Rose

Computer: No, roses are red. Violets are blue.
What color are roses?
(a) Red
(b) Blue
(c) Violet
(d) None of the above

.
.
.

Five years ago most of the work being done in CAI utilized procedures not radically different from the example in the parody. And while there were exceptions (Bitzer's physics inquiry problem and Suppes' logic laboratory, for example), typical instructional interactions were shallow in at least two respects:

(1) the computer always asked the questions, and (2) when the computer responded to the student, the answer did not depend, at least very much, on what had happened before. Thus, whatever might be claimed about its instructional benefits, the stereotype generally lacked imagination and intellectual depth.

The evolution of teaching machines will depend upon progress in artificial intelligence research generally. But, quite apart from these more remote prospects, it seemed possible to improve computer teaching capabilities by taking fuller advantage of *current* computer capabilities. In particular, we sought to develop a computer system that could serve the student as a guide and critic as he attempted to solve a complex problem. We planned to do this by overcoming the two barriers mentioned above.

We permitted the questioning to be initiated by either the student or the computer at various times during their discourse and we enabled the computer's questions and answers at any point to be conditional on arbitrary details in the history of the interaction up to that point. The resulting system, called the Socratic System, is illustrated next in terms of an early application to a problem in clinical medicine.

An Example of Socratic-System Discourse

The student's inputs are indented and capitalized to distinguish them from the computer's responses, which start at the margin and have normal capitalization. Editorial comments on this interaction are enclosed within parentheses.

(At the start of this study session, the computer presents to the student some introductory material on this problem and then tells him to proceed.)

Late this afternoon, a patient was brought to the hospital... . Begin the physical examination as soon as you are ready.

PHYSICAL EXAM.

Fine. What in particular would you like to know?

RESPIRATORY RATE.

It is a common practice in the medical profession to begin an examination by checking the patient's general appearance.

GENERAL APPEARANCE.

Acutely prostrated, lying in bed on his right side. Looks acutely ill, sweaty, apprehensive, but alert. Paroxysmal cough, bringing up a little rusty streaked sputum. Breathing with difficulty.

RESPIRATORY RATE.

20 a minute and shallow.

COMPUTER, ARE YOU A DOCTOR?

I don't recognize what you just typed. Try again.

(This is one of several stock responses to input not recognized by the computer.)

CUOGH?

Do you mean cough?

(This illustrates the recognition of misspelled words through phonetic coding.)

YES.

Yes, cough developed yesterday.

LABORATORY REPORTS?

Finish the physical exam first.

BLOOD PRESSURE?

150/84.

CONTACT WITH BIRDS?

None, except patient has pet canary.

PSITTACOSSIS.

There's insufficient evidence for any conclusive diagnosis at this time. Show a little more restraint please.

PULSE?

100 a minute. Occasional extra systole.

TEMPERATURE?

103

PNEUMOCOCCAL PNEUMONIA.

Stop guessing and continue with the examination.

(This response would be keyed by any diagnosis at this point in the discourse. Note that it is different from the earlier response to an early diagnosis.)

. . .

. . .

LABORATORY REPORTS?

Fine. Which lab tests would you like to have done?

X-RAY.

X-rays will be taken.
Results will be sent to you as soon as possible.

(The computer is deferring a direct response to this request. It will volunteer the information later when "it is available.")

URINE EXAM.

The analysis will be done.

RECTAL EXAM.

You're not being careful. The patient is too acutely ill for a rectal examination.

WBC.

The white count will be taken.

. . .

. . .

PULSE?

Unchanged.
Some of the first series of lab reports have just come in.
Urine exam: insufficient fluid for analysis.
RBC: 5 million.
WBC: 15500.
Differential: Shift to left.
When the other reports are in you'll be notified.

X-RAY?

Results are not available yet.

(Part of the deferred material is volunteered by the computer and some is still deferred.)

. . .

. . .

Perhaps you think it too early to arrive at any definitive conclusion. On the other hand, you may be prepared to make the diagnosis now. Are you?

NO.

Fine. Nevertheless we've got a fair amount of information - at least enough to raise some, perhaps several, diagnostic possibilities. Have you any in mind?

(The computer has now reversed roles with the student and becomes the questioner for a while.)

ACUTE PYELONEPHRITIS.

Unexplained fever, chills, malaise, and flank pain are certainly consistent with acute pyelonephritis. Do you think that the absence of dysuria reinforces the diagnosis?

NO.

Right. And isn't the white count somewhat high for this diagnosis?

NO.

You're wrong. The white count is high, though a later reading is indicated for substantiation.
We've considered one possibility. Do you have another diagnosis in mind?

YES.

Which one?

PNEUMOCOCCAL PNEUMONIA.

Really? Don't you think that tachycardia, dyspnea, and cyanosis would tend to rule out pneumococcal pneumonia?

YES.

You're way off base. There's not a grain of truth in what I just said. You'd better review some elementary medicine, doctor.

. . .

. . .

(The student finishes what he can do on one "day" and is now told that time has passed. It is the "next morning." Some deferred information is finally given to him.)

There was some trouble at the X-ray lab - please excuse the delay. The X-rays are normal. Lateral: no evidence of fluid, no evidence of consolidation.
Now let's see how the patient is today.
Begin the physical when you're ready.

GENERAL APPEARANCE?

Essentially unchanged since last night.

RESPIRATORY RATE?

24 a minute and still shallow.

APPENDICITIS.

Sounds good. But what about the history of cough and the chest signs? What evidence would help you establish this diagnosis?

(Diagnoses are no longer scoffed at in this context. The student has obtained a lot of information.)

EVIDENCE FROM ABDOMINAL EXAM.

Right! Go to it.

. . .

. . .

APPENDICITIS.

Appendicitis? Nonsense. By this time you should have your attention focussed on a different part of the patient's anatomy.

RBC?

An RBC is not indicated now. It's an unnecessary expense for a poor patient.

WBC?

22500.

PLEURAL EFFUSION.

A good contender, but let's be careful.
What evidence would rule it in or out?

POSITIVE EVIDENCE OF ANOTHER DISEASE.

Is there such evidence?

YES.

Well, when you're ready to make that diagnosis, let me know.

TB PNEUMONIA.

Any other thoughts as to diagnosis? The sudden high rise in WBC is against it and you don't have tubercle bacilli in the sputum.

PNEUMOCOCCAL PNEUMONIA.

I see you're still interested in this diagnosis.
Are you absolutely sure?

NO.

What would help you nail down the diagnosis?

EVIDENCE FROM X-RAY FINDINGS.

Good. Anything else?

EVIDENCE FROM BLOOD CULTURE.

What you really need is evidence from the sputum smear and culture. Anyway, continue with your investigation and we'll see what happens.

(The computer is acting as advisor and mentor to help the student come to a correct diagnosis for the right reasons.)

. . .
. . .

The Mentor System

The Socratic System demonstrated that an interesting class of problems -- complex case studies -- could be programmed as tutorial dialogues. Since the system had to be programmed in machine language, it was directly accessible only to knowledgeable computer programmers. The Mentor System was designed to facilitate the description of such complex programmed problems. It embodies and extends the Socratic System facilities for describing problem dialogues. Instructional interactions are described to the Mentor System in terms of the *Mentor Language*. The system compiles this information into a program for carrying out the prescribed instruction in subsequent interaction with a student.

The basic constructs in the Mentor Language are statements of the general form: "*If* the student makes the following input (or one of a set of indicated inputs), *then* if the following conditions (describing some state of the student's information from a relevant sequence of previous questions or assertions that he did or did not carry out) are satisfied, take the following actions (actions can include typing out a message, recording a condition, and changing the context of the discourse)." In its simplest

form, a Mentor statement asserts that a particular student input will cause a specified output to be made by the computer, unconditionally.

The overall program is called a *text*. The statements composing the text are organized into *contexts*. At any point during the dialogue, the interaction is conducted in some particular context, containing the currently relevant inputs and statements. The following example is included to illustrate the use of Mentor language statements. A detailed description of the Mentor Language is given in Appendix 1.1.

An Example of a Mentor Script

This sample of a Mentor script is taken from the Mentor program associated with the medical study case shown above. The parenthesized line numbers at the left are not part of the language, but are given for reference only. Mentor language commands are underscored to distinguish them from English text.

(1) Text Medical Problem.
(2) Start physical examination.
(3) Context physical examination.
(4) Begin → "Late this afternoon, a patient was brought to the hospital... . Begin the physical examination as soon as you are ready."
(5) "Physical exam" → "Fine. What in particular would you like to know?"
(6) "General appearance" (g) →
    1) "Acutely prostrated, lying in bed on his right side... ."
    2) "Unchanged." (M1).

```
(7)  Define diagnosis = ("appendicitis", "psittacosis", ...).
(8)  Any diagnosis → 1)  "There's insufficient evidence ... ."
                     2)  "Stop guessing and continue ... ."
                     3)  "Still guessing?  Finish ... ."
(9)  "Respiratory rate" or "respiration" (r) →
       [g → 1)  "20 a minute and shallow."
            2)  Ml
                Else → "It is common practice in the medical
                       profession ... ."]
          . . .
          . . .

(10) Else → "I don't recognize what you just typed.
             Try again."
(11) End physical examination.
```

The form and meaning of the statements composing the short example are explained now.

Line 1 gives the script the name "Medical Problem."

Line 2 specifies physical examination as the starting context for the interaction.

Lines 3 and 11, respectively, designate the beginning and the end of this context.

Line 4 means that, immediately upon entry into the context named physical examination, the system will respond "Late this afternoon a patient was brought to the hospital ... ."

Line 5 means that, if the current user input is "physical exam," the system will respond "Fine. What in particular would you like to know?"

Line 6 means that, if the current user input is "general appearance" (abbreviated as g), the system will first respond "Acutely prostrated, lying in bed on his right side ... "; subsequently it will respond "Unchanged." (abbreviated as M1).

Line 7 defines diagnosis to be the following set of user inputs: "appendicitis," "psittacosis," ... .

Line 8 means that, if the current user input is any member of the set diagnosis, the system will first respond "There's insufficient evidence ..."; the second time the system will respond "Stop guessing and continue..."; subsequently, it will respond "Still guessing? Finish ... ."

Line 9 means that, if the current user input is either "respiratory rate" or "respiration" (abbreviated as r), then: if the user has previously made an input whose abbreviation is g (standing for "general appearance"), the system will first respond "20 a minute and shallow", subsequently it will respond with the message abbreviated M1 (i.e., with "Unchanged."); else (meaning "in all other cases") the system will respond "It is common practice ... ." The effect of this conditional statement is to require the user to inquire about the patient's general appearance before he will be given the patient's respiration.

Line 10 means that, if the current user input is none of the ones specified in the above statements, the system will respond "I don't recognize what you just typed. Try again." By always concluding a context block with such an else statement we guarantee a response by the system.

Line 11 marks the end of the context block.

Mentor Applications

While Mentor has the limitations common to all contemporary CAI systems based on programmed questions and answers, it is particularly suited for use with programmed problem situations in which the student's procedures are elaborate and complex, yet can be adequately expressed in an explicit logical framework. Mentor has proven to be a useful vehicle for expressing discursive interactions whose development is greatly dependent on the student's questions and answers along the way. Practical applications of this kind can be made in several professional and technical subjects, particularly those in which detailed individual case studies are used as an important means of imparting knowledge and skill. Three examples of such applications of Mentor to problems in management, educational administration, and criminal investigation are given in Appendix 1.2. (The last one, which is a parody of the naive mystery story, was designed as a demonstration of the use of Mentor language and is discussed in Appendix 1.1.) Mentor has also been applied to clinical medical study, as in the example shown above, and to electronic trouble-shooting problems.

3. Stringcomp: The Use of a Programming Language for Instructional Applications

At the outset of this project, most of the computer-assisted instructional programs under active development were being written in terms of special instructional languages, such as Computest and Coursewriter. These languages permit the expression of programmed questions and answers. Since instructional material was to be written by persons not necessarily interested in learning computer programming, the use of a programming language, such as Fortran, did not seem generally feasible. Around this time, new programming languages such as the JOSS language*, expressly designed to make programming easily accessible to scientists, engineers, and other intelligent noncomputer-experts, came into being. We were interested in exploring the use of programming languages as instructional tools, and the appearance of these simple new languages made this interest timely.

To begin with, programming languages might not seem particularly well suited to the expression of programmed drills, lessons, or problems. Programming languages usually deal with numerical calculation or with text manipulation and seldom handle both areas well. On the other hand, if a simple programming language effectively permitted the expression of numerical and formal calculation procedures as well as text or character-string operations, it could enhance and extend the programmed question and answer capabilities of the standard specialized instructional languages.

---

* Shaw, J.C. "JOSS: A Designer's View of an Experimental On-Line Computing System," AFIPS Conference Proc. Fall Joint Computer Conference, Spartan Press, 1964.

Our object was to investigate the instructional uses of a programming language that would join the simplicity of JOSS for expressing numerical calculations with the straightforward features present in string-manipulation languages, such as COMIT and SNOBOL, for analyzing and constructing text. To do this required designing such a language. We started with *Telcomp**, our existing JOSS dialect. Telcomp had already proven easy to teach to persons familiar with arithmetic. We augmented Telcomp with facilities for processing arbitrary strings of information, numerical and non-numerical, and called the resulting programming language *Stringcomp*.

Before showing the various ways in which Stringcomp was used for instruction, we give a brief description of this new language. A detailed description of Stringcomp is included in Appendix 2.

3.1 Telcomp and Stringcomp

Telcomp is a language for manipulating numerical expressions. A Telcomp expression can be an integer, a decimal number, a floating point number, a numerically-valued variable, or an arithmetic expression in any of those. Variables are designated by English either strings such as X or VOLUME. Subscripted variables or arrays are designated by setting off the indices with brackets, e.g., X[7], VOLUME[X,Y,Z].

Addition, subtraction, multiplication, division, and exponentiation are designated, respectively, by +, -, *, /, and ↑.

---

* Myer, Theodore, "Telcomp Manual" Cambridge, Mass., Bolt Beranek and Newman Inc., 1966.

To perform these operations at a teletypewriter terminal, one uses the TYPE operation. Thus, if a user types TYPE 2 + 3 and presses the carriage return key, Telcomp types back: 2 + 3 = 5. It then carriage-returns and types the mark ← which indicates that it is ready for another instruction from the user.

Similarly, the following inputs (shown on the left) will cause Telcomp to make the corresponding outputs (shown on the right).

Input	Telcomp Output
←TYPE 2-3	2-3 = -1
←TYPE 2*3	2*3 = 6
←TYPE 2/3	2/3 = .66666667
←TYPE 2↑3	2↑3 = 8

These operations may be compounded and parentheses can be used to indicate grouping Thus, TYPE 2.75 + 3↑(2-3+4) causes Telcomp to type 2.75 + 3↑(2-3+4) = 29.75. Several operations can be requested on a single input. Thus, the input TYPE 2-3, 2*3, 2/3, 2↑3 will result in all of the outputs shown above.

Variables are assigned values by the SET operation, as follows:

Input	Telcomp Output
←SET X=4	(No output)
←TYPE X*(X-1)	X*(X-1)=12
←SET Y=X↑2, A=3*Y	(No output)
←TYPE X,Y,A	X=4, Y=16, A=48
←SET X=1+X	(This means replace the value of X by 1 plus its current value. The equal sign in a SET command denotes replacement.)
←TYPE X	X=5

Input	Telcomp Output
←SET LENGTH=5, WIDTH=17.1	
←SET AREA=LENGTH*WIDTH	
←TYPE AREA	AREA = 85.5

As well as arithmetic operations, Telcomp expressions can include functions such as square root, exponential, natural logarithm, logarithm (base 10), sin, cos, etc., etc. The following Telcomp instructions illustrate the use of the first three of these functions.

```
←SET X=6
←TYPE X, SQRT(X+1), EXP(X), LN(2.7), EXP(LN(X))

                        X=6
                        SQRT(X+1) = 2.645751
                        EXP(X) = 403.4287
                        LN(2.7) = .9932518
                        EXP(LN(X)) = 6
```

The last example shows how functions can be combined in Telcomp.

The functions MAX(A,B, ...) and MIN(A,B, ...) determine the maximum and the minimum of the values of the arguments enclosed in parentheses. The functions IP(A) and FP(A) evaluate the integer part and fractional part of the argument, respectively. Thus,

```
←SET V=MAX(3↑2,SQRT(103),12.423)
←TYPE V, IP(V), FP(V)
```

causes Telcomp to type

```
V = 12.423
IP(V) = 12
FP(V) = .423
```

Telcomp treats the conditional operation IF as a statement modifier. It interprets the relations <, =, > in the usual way as less than, equal to, and greater than, respectively. Thus,

```
←SET X=2, Y=3
←TYPE X↑Y IF Y>X
←TYPE X-Y IF X-Y>Ø
```

causes Telcomp to type

```
X↑Y = 8
```

It does not type the value of X-Y since X-Y is not positive.

Telcomp Programs

A Telcomp command with a number in front of it is called a STEP. A STEP is numbered with an integer (called a PART number) followed by a decimal. STEPS are grouped together in PARTS. Thus, STEPS 4.1, 4.2, and 4.3 make up PART 4 in the following Telcomp program.

```
←4.1  SET ZEBRA = A
←4.2  TYPE B↑(ZEBRA/2)
←4.3  TO STEP 2.5 IF A<1Ø
```

(The last command is a TO command and directs Telcomp to find its next command at STEP 2.5, whatever that is, if the value of A is less than 1Ø. TO commands are used to change the order in which Telcomp carries out commands.)

Telcomp cannot carry out a program until it is given a DO command. Thus if A=6, and B=5, the command

```
←DO PART 4
```

will cause steps 4.1, 4.2, and 4.3 to be performed, causing Telcomp to type

B↑(ZEBRA/2) = 125
and then to find its next command at STEP 2.5 in PART 2.

One other command can be used to set values of Telcomp variables. The command DEMAND causes a program to wait for the user to type in a requested value, as in the following example.

```
←1.1  DEMAND A
←1.2  SET C=A + SQRT(A)
←1.3  TYPE C
←DO PART 1
```

The DO command carries out STEP 1.1. This causes Telcomp to type
A =
and wait for the user to respond. If the user types in 16 (followed by a carriage return), Telcomp will carry out STEP 1.2 and STEP 1.3, causing the typeout
C = 2Ø.

The command modifier FOR is used to successively set a variable to each of a range of values. Thus,
←TYPE X↑2 FOR X=3, 4, 7, 9
causes Telcomp to type
X↑2 = 9
X↑2 = 25
X↑2 = 49
X↑2 = 81
An equivalent command is
←TYPE X↑2 FOR X=3:2:9
In this command, FOR X=3:2:9 means for X=3 in steps of 2 until X exceeds 9.

The FOR modifier is often used to call programs as shown in the following example, which computes the first two Pythagorean triples.

```
←2.1  SET C = SQRT(A↑2 + B↑2)
←2.2  TYPE A,B,C IF FP(C) = Ø
←DO PART 2 FOR A=1:1:8 FOR B=1:1:8
```

This program causes Telcomp to type

```
A = 3
B = 4
C = 5

A = 6
B = 8
C = 1Ø
```

The IF condition in STEP 2.2 asserts, in effect that C must be an integer (i.e., its fractional part must be zero).

From Telcomp to Stringcomp

Telcomp can be used to type out any string of characters. Thus, TYPE "HELLO THERE 75249X!" causes Telcomp to type the string enclosed in quotes. But, Telcomp cannot interpret a string typed in by a user unless it can be evaluated numerically. If a user typed in "HELLO THERE 75249X" in response to a DEMAND instruction, Telcomp could not interpret the input and would issue an error comment. To permit Stringcomp to interpret strings that do not have a numerical value (as well as strings that have a numerical value but which we want to evaluate formally or symbolically rather than numerically) requires several additions to the Telcomp language. These include a new variable type, a *string variable* and some new operators, functions, and Boolean relations.

A string variable in Stringcomp is one whose value is a string of characters, numerical and/or non-numerical. A string variable is given a value by the ordinary Telcomp SET statement. For example,

SET X = "AB"

sets the variable X to the value of the literal string AB.

SET X = Y

sets the variable X to the value of the variable Y, whatever that happens to be.

SET X = "AB".Y

(where the "." denotes concatenation) sets the variable X to the concatenation of the literal string AB and the value of Y. Thus, if Y = "13.2", the result of the SET instruction is to give X the value "AB13.2".

The value of a numerical string can be used as a number in a calculation. If W = "12", a (numerical) string, then in executing the instruction A = W + 1, W is converted from a string to a number and A is given the numerical value 13.

A Boolean valued function NUM(X) can be used in an IF clause to test if the value of the variable X can indeed be interpreted as a number.

Several functions are provided in Stringcomp for testing and analyzing strings. For example,

STL(X) gives the length (the number of characters) in the string X. STL("JOE") = 3.

LOC(X,Y) gives the location of the first character of the string Y in the string X if Y is contained in X; otherwise it gives the answer ∅.
LOC("MOZART", "ART") = 4
LOC("MOON", "NO") = ∅

EXT(X,Y,Z) gives the Yth through Zth characters in string X.
EXT("SUNDAY", 1,3) = SUN.

COND(C1,V1,C2,V2,...) gives V1 if C1 is true; else it gives V2 if C2 is true, and so on.
COND(X>Y, X, X<Y, Y, X=Y, Z) gives X if X is greater than Y, Y if X is less than Y, and Z if X and Y are equal.

These functions can be compounded. Thus, if A = "SOUP", EXT(A,LOC(A,"U"),STL(A)) = UP.

To check that a string has a prescribed structure or to find specified substrings, the comparison operator IS may be used. This operator is used in an IF clause, in the form IF X IS Y, where X is any string expression and Y is any *pattern*. A pattern is either an elementary pattern or a concatenation of elementary patterns. The elementary patterns include:

(1) Literal strings such as "ABC".
(2) String variables as defined above.
(3) #n, where n is an integer, denoting any string segment of length n.
(4) #, denoting any string segment whatever.

The form P1/P2/.../Pn denotes any one among the elementary patterns P1, P2,..., Pn.

In (3) and (4), #n[V] and #[V] set the value of the variable V to the indicated string segment (but only if the entire string X matches the entire pattern Y in the IF clause).

The following Stringcomp program illustrates the use of the IS operator. The program tests whether any string S is symmetric, i.e., whether it is a palindrome (such as "MADAM" or "ROTOR" or "BOB" or "BOOB").

```
1.Ø  TYPE "PLEASE TYPE IN A STRING."
1.1  DEMAND S
1.2  TO STEP 1.2 IF S IS #1[X].#[S].X
1.3  TYPE "YOUR STRING IS NOT SYMMETRIC." IF STL(S)>1
1.4  TYPE "YOUR STRING IS SYMMETRIC." IF STL(S)<=1
```

Step 1.Ø requests the user to input a string. Step 1.1 names the requested string S and waits for S to be typed in. Step 1.2 tests whether the first and last characters of S are identical. If they are, this instruction renames the middle part of the string S and repeats the test on this new S. When the test fails, steps 1.3 and 1.4 are performed. If the final S is more than one character long, the given string was unsymmetric; otherwise it was symmetric. The appropriate message is typed in either case.

3.2 Instructional Applications of Stringcomp

Stringcomp has been used as an instructional tool in four distinct kinds of exploratory applications -- programmed teaching, diagnostic testing, student programming, and instructional monitoring. The instructional contexts have comprised topics

from mathematics, physics, and other scientific and technical subjects. Illustrative examples follow.

3.2.1 Stringcomp in Programmed Teaching and Testing

An algebraic expression such as AX↑2 - BX + C can be typed into Stringcomp and interpreted in any of three ways, as:
(1) a message text to be matched with a set of anticipated answers to a question;
(2) a numerical expression to be evaluated, given that known numerical values for A, B, C, and X are available;
(3) a formal string to be matched with a set of patterns to determine its structure and content.

The simplest example of the last mode of manipulation is finding whether a string contains a given substring. This is often used in CAI lessons to find whether an expected answer is contained anywhere in a student's response. More elaborate pattern matching and string manipulation procedures coupled with numerical computation procedures can be used to write programs that extend and enhance the art of instructional programming. To show this, we have designed and programmed several Stringcomp exercises and lessons, mainly in the area of elementary mathematics.

The six algebra transcripts that follow required relatively straightforward and unsophisticated Stringcomp programming. They illustrate some of the effects obtainable from using an appropriate programming language and are merely suggestive of the potential of this line of approach. In these transcripts and those that follow, the student's inputs are underscored to distinguish these from the computer's outputs.

ALGEBRA LEARNING TOOL 1

This program is the first of a series of programs referred to as Algebra Learning Tools. The exercises were designed to supplement regular classroom instruction in algebra. The first program is designed to aid in the transition of thinking in terms of equations with fixed numerical coefficients (3X+4=7) to equations in variable coefficients (AX+B=C).

The program approaches this problem in three stages. The first stage presents a series of exercises which the pupil is asked to solve with an eye open for any patterns that proved helpful in solving them. If he gets any of the problems wrong, the computer, without any explanation, tells him that his answer made the right side of the equation too big or too small.

In the second stage the student is asked if he sees any pattern that would help him to get closer to the right answer with every guess. If yes, he is tested on this part. If no, he is given more problems and asked again later on.

In the third stage the program asks whether or not the student has discovered a way to get the correct answer every time, i.e., if he knows the general solution.

WHILE YOU DO THE FOLLOWING PROBLEMS, TRY TO DISCOVER ANY PATTERNS THAT MIGHT MAKE SOLVING THEM EASIER.

5X - 12 = 8
X= 3

NO,
5 TIMES 3, MINUS 12, DOES NOT EQUAL 8
IT EQUALS 3

8 IS GREATER THAN 3.

5X - 12 = 8
X= 4

RIGHT!
5 TIMES 4, MINUS 12, EQUALS 8

7X - 25 = -18
X= 1

RIGHT!
7 TIMES 1, MINUS 25, EQUALS -18

7X - 16 = 12
X= 4

RIGHT!
7 TIMES 4, MINUS 16, EQUALS 12

4X - 12 = -8
X= 1

RIGHT!
4 TIMES 1, MINUS 12, EQUALS -8

2X - 2 = Ø
X= 1

RIGHT!
2 TIMES 1, MINUS 2, EQUALS Ø

DO YOU SEE A PATTERN THAT WOULD ENABLE YOU TO GET THE RIGHT ANSWER EVERY TIME? NO

DO YOU SEE A PATTERN THAT WOULD HELP YOU TO GET CLOSER TO THE RIGHT ANSWER EVERY TIME? YES

LET'S SEE IF YOU ARE RIGHT.

4X - 1 = 3
X= 3

NO,
4 TIMES 3, MINUS 1, DOES NOT EQUAL 3
IT EQUALS 11

11 IS GREATER THAN 3.

4X - 1 = 3
X= 1

RIGHT!
4 TIMES 1, MINUS 1, EQUALS 3

9X - 13 = 23
X= 4

RIGHT!
9 TIMES 4, MINUS 13, EQUALS 23

1X - 9 = -4
X= 5

RIGHT!
1 TIMES 5, MINUS Ø, EQUALS -4

I THINK YOU KNOW HOW TO IMPROVE YOUR GUESSES.

DO YOU SEE A PATTERN THAT WOULD ENABLE YOU TO GET THE RIGHT ANSWER THE FIRST TIME, EVERY TIME? YES

LET'S SEE IF YOU ARE RIGHT.

9X - 7 = 2
X= 1

RIGHT!

7X - 7 = 28
X= 5

RIGHT!

```
7X - 8 = -1
X= 1

RIGHT!

1ØX - 16 = 14
X= 3

RIGHT!

5X - 13 = 2
X= 3

RIGHT!
VERY GOOD, YOU'VE GOT IT!

←
```

ALGEBRA LEARNING TOOL 2

This second Algebra Learning Tool treats linear equations starting with the problem X+A=B and working up to AX+B=C. The student works with a number of problems with numerical coefficients, as in the previous program, before tackling the case with variable coefficients. The solution (C-B)/A is asked for specifically, and the program is able to identify certain standard variations of this solution.

The coefficients are varied systematically from problem to problem to help the student to draw the proper conclusions. The student may ask for explanatory text if he does not understand any of the questions.

IF YOU DON'T UNDERSTAND ANY OF THE QUESTIONS, TYPE '?' AND ENTER.

CAN YOU SOLVE THE EQUATION X+A=B FOR X IN TERMS OF A AND B? ?

IN THE EQUATION X+A=B, A,B, AND X STAND FOR 'ANY NUMBER'.
TRANSLATED INTO ENGLISH IT SAYS:
    SOME NUMBER CALLED 'X', WHEN ADDED TO ANOTHER
    NUMBER CALLED 'A', GIVES THE SUM 'B'.
IN THIS CASE I'VE ASKED YOU TO ASSUME THAT YOU KNOW THE VALUES OF A AND B, BUT DO NOT KNOW WHAT X IS. CAN YOU TELL ME WHAT X IS FOR ANY NUMBER A AND ANY NUMBER B? IF NOT TYPE 'NO' WHEN I RE-ASK THE QUESTION AND WE'LL TRY SOME PROBLEMS WITH SPECIFIC VALUES FOR A AND B.

CAN YOU SOLVE THE EQUATION X+A=B FOR X IN TERMS OF A AND B? YES

X + A = B X= B+A
NO, THAT'S NOT RIGHT.

WHY DON'T WE TRY AGAIN AFTER A FEW PROBLEMS.

DO THE FOLLOWING PROBLEMS KEEPING AN EYE OUT FOR A SOLUTION TO THE GENERAL PROBLEM, X+A=B.

X + 1 = 1 X= 1
NO, 1 PLUS 1 DOES NOT EQUAL 1
IT EQUALS 2

X + 1 = 2 X= 1
RIGHT!
1 PLUS 1 EQUALS 2

X + 1 = 3 X= 2
RIGHT!
2 PLUS 1 EQUALS 3

X + 1 = 4 X= 3
RIGHT!
3 PLUS 1 EQUALS 4

X + 1 = 15 X= 14
RIGHT!
14 PLUS 1 EQUALS 15

X + 3 = 15 X= 13
NO, 13 PLUS 3 DOES NOT EQUAL 15
IT EQUALS 16

```
X + 5 = 15     X= 1Ø
RIGHT!
1Ø PLUS 5 EQUALS 15

X + 7 = 15     X= 8
RIGHT!
8 PLUS 7 EQUALS 15

X + -8 = 13     X= 21
RIGHT!
21 PLUS -8 EQUALS 13

X + -13 = 13     X= 25
NO, 25 PLUS -13 DOES NOT EQUAL 13
IT EQUALS 12

X + -18 = 13     X= 31
RIGHT!
31 PLUS -18 EQUALS 13

X + -23 = 13     X= 36
RIGHT!
36 PLUS -23 EQUALS 13

DO YOU THINK NOW THAT YOU COULD SOLVE X+A=B FOR X IN TERMS
OF A AND B?  YES

X + A = B     X= B-A
RIGHT!

CAN YOU SOLVE THE EQUATION AX+B=C FOR X IN TERMS OF A,B, AND C? ?

THIS IS LIKE THE PROBLEM YOU HAVE ALREADY SOLVED, X+A=B, EXCEPT
THAT IN THIS CASE THE UNKNOWN ('X') IS MULTIPLIED BY SOME NUMBER
'A' WHICH WHEN ADDED TO 'B' GIVES 'C'.  AGAIN I WANT X'S VALUE
WITH RESPECT TO THE KNOWN CONSTANTS, A,B, AND C.  IF YOU STILL
HAVE TROUBLE TYPE 'NO' AND I'LL GIVE YOU A FEW PROBLEMS WITH
SPECIFIC A'S, B'S AND C'S TO HELP YOU ALONG.

CAN YOU SOLVE THE EQUATION AX+B=C FOR X IN TERMS OF A,B, AND C?
YES

AX + B = C     X= B-C/A
NO, I DON'T THINK THAT'S RIGHT.

WHY DON'T YOU TRY AGAIN AFTER A FEW PROBLEMS.
```

```
2X + 2 = 1     X= -1/2
RIGHT!
2 TIMES -1/2 PLUS 2 EQUALS 1

2X + 2 = 2     X= 1
NO, 2 TIMES 1 PLUS 2 DOES NOT EQUAL 2
IT EQUALS 4

2X + 2 = 3     X= 1/2
RIGHT!
2 TIMES .5 PLUS 2 EQUALS 3

3X + -2 = 6    X= 8/3
RIGHT!
3 TIMES 8/3 PLUS -2 EQUALS 6

3X + -3 = 6    X= 3
RIGHT!
3 TIMES 3 PLUS -3 EQUALS 6

3X + 6 = 23    X= 2/3
NO, 3 TIMES 2/3 PLUS 6 DOES NOT EQUAL 23
IT EQUALS 8

4X + 6 = 23    X= 2/3
NO, 4 TIMES 2/3 PLUS 6 DOES NOT EQUAL 23
IT EQUALS 8.666667

5X + 6 = 23    X= 2/5
NO, 5 TIMES 2/5 PLUS 6 DOES NOT EQUAL 23
IT EQUALS 8

6X + 6 = 23    X= 29/6
NO, 6 TIMES 29/6 PLUS 6 DOES NOT EQUAL 23
IT EQUALS 35

7X + 6 = 23    X= 17/7
RIGHT!
7 TIMES 17/7 PLUS 6 EQUALS 23

8X + 6 = 23    X= 17/8
RIGHT!
8 TIMES 17/8 PLUS 6 EQUALS 23

9X + 6 = 23    X= 17/9
RIGHT!
9 TIMES 17/9 PLUS 6 EQUALS 23
```

```
NOW DO YOU THINK YOU CAN SOLVE AX+B=C FOR X? NO
CAN YOU SOLVE IT FOR 'AX' IN TERMS OF B AND C? ?

THIS IS ALMOST EXACTLY LIKE X+A=B EXCEPT THAT INSTEAD OF 'X' WE
HAVE 'AX'.  YOU CAN THINK OF THE PROBLEM AS Y+B=C WHERE Y=AX.
ALL YOU HAVE TO DO IS TO SOLVE FOR Y IN TERMS OF B AND C.

CAN YOU SOLVE IT FOR 'AX' IN TERMS OF B AND C?  YES

AX + B = C     AX= C-B
RIGHT!

NOW DO YOU THINK YOU CAN SOLVE AX+B=C FOR X? YES

AX + B = C     X= C/A-B/A
RIGHT!

DO YOU SEE HOW YOU COULD SAY THAT MORE SIMPLY? NO

TRY SOLVING FOR AX FIRST, THEN SOLVING FOR X.
AX + B = C     X= C-B/A
DO YOU MEAN C-(B/A)? NO
WHAT DO YOU MEAN? C-B/A

IF THAT WERE TRUE, THEN (SUBSTITUTING YOUR ANSWER FOR 'X')
A(C-B/A) + B = C
AND
AC -B + B = C
AND
AC WOULD EQUAL C
SINCE THAT CAN ONLY BE TRUE WHEN A=1 OR C=Ø, YOU MUST HAVE GOOFED.
BUT SINCE YOU WERE SO CLOSE, TRY AGAIN.

AX + B = C     X= (C-B)/A
RIGHT!

←
```

ALGEBRA LEARNING TOOL 3

These exercises are similar to the first set except that the program directs the student to use the results from previous guesses in making his current guess.

```
IN THIS PROGRAM YOU AND I WILL WORK TOGETHER IN TRYING TO GUESS
THE SOLUTIONS TO LINEAR EQUATIONS.

2X + 2 = 22

SHOULD I TRY -2? YES
NO,
2 TIMES -2, PLUS 2, DOES NOT EQUAL 22
IT EQUALS -2
22 IS GREATER THAN -2.

SHOULD THE NEXT TRY BE MORE OR LESS? MORE

HOW MUCH? 5
NO,
2 TIMES 3, PLUS 2, DOES NOT EQUAL 22
IT EQUALS 8
22 IS GREATER THAN 8.

SHOULD THE NEXT TRY BE MORE OR LESS?  MORE

HOW MUCH? 5
NO,
2 TIMES 8, PLUS 2, DOES NOT EQUAL 22
IT EQUALS 18
22 IS GREATER THAN 18.

SHOULD THE NEXT TRY BE MORE OR LESS? MORE

HOW MUCH? 2

IT WORKS!
2 TIMES 1Ø, PLUS 2, EQUALS 22

WOULD YOU LIKE ME TO DO ANOTHER? YES

1ØX + 9 = 59

SHOULD I TRY -4? YES
NO,1Ø TIMES -4, PLUS 9, DOES NOT EQUAL 59
IT EQUALS -31
59 IS GREATER THAN -31.

SHOULD THE NEXT TRY BE MORE OR LESS? MORE

HOW MUCH? 1Ø
```

```
NO,
1Ø TIMES 6, PLUS 9, DOES NOT EQUAL 59
IT EQUALS 69
69 IS GREATER THAN 59.

SHOULD THE NEXT TRY BE MORE OR LESS? LESS

HOW MUCH? 1

IT WORKS!
1Ø TIMES 5, PLUS 9, EQUALS 59

WOULD YOU LIKE ME TO DO ANOTHER? NO

WOULD YOU LIKE TO TRY ONE BY YOURSELF? YES

5X + 14 = 34
WHAT IS YOUR GUESS? 5
NO,
5 TIMES 5, PLUS 14, DOES NOT EQUAL 34
IT EQUALS 39
39 IS GREATER THAN 34.

SHOULD THE NEXT TRY BE MORE OR LESS? LESS

HOW MUCH? 1

IT WORKS!
5 TIMES 4, PLUS 14, EQUALS 34
```

. . .

ALGEBRA LEARNING TOOL 4

This program asks a student to express a straightforward algebra word problem stated in English as a formal equation, and guides him in analyzing his difficulties.

```
SOME NUMBER WHEN DIMINISHED BY -4 THEN MULTIPLIED BY 1Ø EQUALS 7.
WHAT IS THE NUMBER? 4

THAT WASN'T RIGHT.

PLEASE TYPE THE EQUATION YOU USED IN ATTEMPTING TO SOLVE THIS
PROBLEM LETTING 'X' EQUAL THE UNKNOWN.

1ØX-4+7
THERE'S NO '=' SIGN IN THAT!

WILL YOU TRY AGAIN? YES

WHAT IS THE EQUATION? 1ØX--4=/

NO WONDER YOU GOT THE WRONG ANSWER, THAT'S NOT THE RIGHT EQUATION!

WHAT YOU DID WAS TO MULTIPLY BY 1Ø BEFORE SUBTRACTING -4
TRY AGAIN.

WHAT IS THE EQUATION? 1Ø(X+4)=7

THAT LOOKS ALRIGHT.

TRY THE FIRST PROBLEM AGAIN.

WHAT IS THE NUMBER? 3

YOU MUST HAVE MADE AN ERROR SOLVING IT.

SOLVE THIS FOR X
X - -4 = 7
X= 3

OK, TRY THIS ONE:
1ØX = 7
X= 7/1Ø

VERY GOOD, NOW TRY THIS:
1Ø(X--4)=7
X= 7/1Ø-4
RIGHT!
YOU HAVE JUST SOLVED THE ORIGINAL PROBLEM!
WOULD YOU LIKE TO TRY A NEW ONE? YES
SOME NUMBER WHEN DIMINISHED BY Ø THEN MULTIPLIED BY 7 EQUALS 1.
WHAT IS THE NUMBER? 1/7
VERY GOOD!
WOULD YOU LIKE TO TRY A NEW ONE? NO
```

ALGEBRA LEARNING TOOL 5

This program is really just a translation of Algebra Learning Tool 3 from linear equations to word problems. The procedure is almost identical: a problem is given and a guess is made and improved upon until the correct answer is found. In this program the computer cycles through eight different types of word problems - each one a little different and usually a little harder than the preceding one - and points out, for each wrong answer, the inconsistencies involved.

The program was designed to remove any hesitancy the student might have about guessing at answers to problems where the correct solution is not immediately obvious and the method of solution even less.

```
THE SUM OF THREE CONSECUTIVE NUMBERS IS 63.
IS THE LARGEST OF THE THREE NUMBERS 24? NO
WHAT IS THE LARGEST NUMBER? 23
BUT 21 + 22 + 23 = 66, NOT 63.
SHOULD THE NEXT TRY BE MORE OR LESS? LESS
HOW MUCH LESS? 1
IS THE LARGEST NUMBER 22? YES
RIGHT!

LET'S DO ANOTHER ONE.
```

ANNE IS 7 YEARS OLDER THAN BILL.
THE SUM OF THEIR AGES IS 9.

IS ANNE 2 YEARS OLD? YES

BY THE FIRST STATEMENT THAT MAKES BILL -5
AND BY THE SECOND, 7 YEARS OLD.
NEGATIVE AGE?
AND HOW CAN BILL HAVE TWO AGES AT ONCE!

SHOULD THE NEXT TRY BE MORE OR LESS? MORE

HOW MUCH MORE? 6

IS ANNE 8 YEARS OLD? YES

RIGHT!

LET'S DO ANOTHER ONE.

FRED IS 8 YEARS OLDER THAN HENRY.
THE SUM OF THEIR AGES IS 16.

IS FRED 2 YEARS OLD? NO

HOW OLD IS FRED? 12

RIGHT!

LET'S DO ANOTHER ONE.

IN A BOX OF 12 PIECES OF FRUIT THERE ARE 1Ø MORE APPLES THAN ORANGES.

ARE THERE 2 APPLES? NO

HOW MANY APPLES ARE THERE? 1Ø

BY THE FIRST STATEMENT THAT MAKES 2 ORANGES
AND BY THE SECOND, Ø.

HOW CAN THERE BE TWO NUMBERS OF ORANGES AT ONCE!

SHOULD THE NEXT TRY BE MORE OR LESS? MORE

HOW MUCH MORE? 1

ARE THERE 11 APPLES? YES

LET'S DO ANOTHER ONE.

IN A BOX OF 14 BOOKS THERE ARE 4 MORE PLAYS THAN POEMS.

ARE THERE 2 PLAYS? NO

HOW MANY PLAYS ARE THERE? 9

RIGHT!

LET'S DO ANOTHER ONE.

ONE DAY BILL BUYS A LARGE APPLE AND A SMALL ONE FOR $Ø.18
THE NEXT DAY HE BUYS 1 LARGE AND 5 SMALL APPLES AT THE SAME RATE FOR $Ø.3Ø
DO THE LARGE APPLES COST 2 CENTS APIECE? NO

HOW MUCH DO THEY COST APIECE? 3

THAT MEANS THAT ON THE FIRST DAY SMALL APPLES COST 15 CENTS AND ON THE SECOND, 27

I SAID THAT HE BOUGHT THEM AT THE SAME RATE!

SHOULD THE NEXT TRY BE MORE OR LESS? MORE

HOW MUCH MORE? 12

DO THE LARGE APPLES COST 15 CENTS APIECE? YES

RIGHT!

LET'S DO ANOTHER ONE.

ONE DAY HENRY BUYS A LARGE ORANGE AND A SMALL ONE FOR $Ø.2Ø
THE NEXT DAY HE BUYS 1 LARGE AND 5 SMALL ORANGES AT THE SAME RATE FOR $Ø.4Ø
DO THE LARGE ORANGES COST 2 CENTS APIECE? NO

HOW MUCH DO THEY COST APIECE? 15

RIGHT!

LET'S DO ANOTHER ONE.

FIVE YEARS AGO SALLY WAS 1/5 AS OLD AS HER MOTHER WAS THEN, AND 5 YEARS FROM NOW SHE WILL BE 3/7 AS OLD AS HER MOTHER WILL BE THEN.

IS SALLY 2 NOW? NO

HOW OLD IS SALLY NOW? 1Ø

RIGHT!

LET'S DO ANOTHER ONE.

IF JOHN GIVES JAMES 6 APPLES, JOHN WILL HAVE 2/3 AS MANY APPLES AS JAMES. IF JAMES GIVES JOHN 9 APPLES, JAMES WILL HAVE 2/3 AS MANY APPLES AS JOHN.

DOES JOHN HAVE 2 APPLES? NO

HOW MANY APPLES DOES JOHN HAVE? 3Ø

BY THE FIRST STATEMENT THAT GIVES JAMES 3Ø APPLES
AND BY THE SECOND, 35.

SHOULD THE NEXT TRY BE MORE OR LESS? MORE

HOW MUCH MORE? 3

DOES JOHN HAVE 33 APPLES? YES

BY THE FIRST STATEMENT THAT GIVES JAMES 34.5 APPLES
AND BY THE SECOND, 37.

SHOULD THE NEXT TRY BE MORE OR LESS? MORE

HOW MUCH MORE? 3

DOES JOHN HAVE 36 APPLES? YES

RIGHT!

. . .

←

ALGEBRA TEST

This program was written to explore the possibilities of administering an algebra test by computer. In this test we dealt with simultaneous equations and tried to define for any problem a series of lesser or subproblems which led up to it. The pupil taking the test starts with a word problem involving simultaneous equations and controls the sequence of questions by his performance on the previous ones.

If the student has difficulties with word problems, he is given numerical problems first involving simultaneous linear equations. If he cannot solve these, he is given single linear equations. In this fashion, the problems get progressively simpler until he has problems he can solve. When the problems can get no simpler or when it becomes clear what the student's weaknesses are, the test ends and the scores are filed by problem type under the pupil's name.

Whenever he can solve problems at a given level he is again given a chance to solve problems at the next higher level of difficulty. In the case of simultaneous linear equations, two lines of attack are tested. If the student cannot solve equations by substitution and elimination, he is given a chance to try subtraction of equations.

The program is designed to accept virtually any student input. Most of the numbers used in the problems are chosen randomly to eliminate the likelihood of cribbing. Other Stringcomp programs were written to assist test designers in constructing and administering programmed tests like the one shown here.

SIMULTANEOUS EQUATIONS

CAN YOU SOLVE SIMULTANEOUS EQUATIONS? YES

WHAT IS YOUR NAME? TEACHR

SOLVE THIS WORD PROBLEM:

FIND TWO NUMBERS WHOSE SUM IS 196 IF THE LARGER IS 44 BIGGER
THAN THE SMALLER.

LARGER NUMBER...100
SMALLER NUMBER...41

ARE YOU SURE? YES

HERE IS A PAIR OF LINEAR EQUATIONS.
WHAT VALUES OF X AND Y WILL SATISFY BOTH OF THEM?
2X + 1Y = 9
3X + 2Y = 16
X = 4
Y = Z

I DON'T UNDERSTAND YOUR ANSWER. REPEAT IT PLEASE.
X = 4
Y = 4
WOULD YOU LIKE TO TRY ANOTHER? NO

SOLVE THIS LINEAR EQUATION FOR X
0X + -5 + -5X = 0X + 0
X = 0
WILL YOU TRY ANOTHER? NO

LET'S TRY AN EASIER ONE.
FIND THE VALUE OF X
-5X + -3 = -3
X = 2
WILL YOU TRY ANOTHER? NO

LET'S TRY SOME ARITHMETIC PROBLEMS.
10 + 2 = 12
4 - 7 = -3
2 * 8 = 16
6 / 9 = 6/9

WHAT NUMBER WOULD YOU ADD TO THE FOLLOWING TO GET ZERO?-
-4...4

IN EACH OF THE FOLLOWING TELL ME WHAT NUMBER WOULD MAKE THE EXPRESSION TRUE.
11X = 33
X = 3
11X = 77
X = 7
4X = 28
X = 7
6X = 24
X = 4

-2X + Ø = 1
X = -1/2

TYPE THE ANSWER IN TERMS OF X
9X + 2X = 11X
9X - 5X = 4X

SOLVE THIS LINEAR EQUATION FOR X
ØX + 5 + 4X = 5X + 3
X = ?

TRY THIS PROBLEM:
IF Z = 7X + 3
AND 5Z + -3 = Ø
THEN X = 2
WILL YOU TRY ANOTHER? NO

CONSIDER THE FOLLOWING PAIR OF LINEAR EQUATIONS
2X + 6Y = 2Ø
1X + 2Y = 8

IF YOU WERE TRYING TO MAKE THE COEFFICIENTS OF THE TWO 'X' TERMS THE SAME, BY WHAT NUMBER (FRACTIONS ACCEPTABLE) WOULD YOU MULTIPLY EACH EQUATION?

THE SECOND? 16
WANT TO TRY ANOTHER? YES
CONSIDER THE FOLLOWING PAIR OF LINEAR EQUATIONS
2X + 1Y = 9
3X + 2Y = 16

IF YOU WERE TRYING TO MAKE THE COEFFICIENTS OF THE TWO 'X' TERMS THE SAME, BY WHAT NUMBER (FRACTIONS ACCEPTABLE) WOULD YOU MULTIPLY EACH EQUATION?
THE FIRST? 3
THE SECOND? 2

IF YOU PERFORMED THE OPERATION YOU HAVE DESCRIBED ABOVE, WHAT
WOULD THE CONSTANT (THE NUMBER AFTER THE '=') BE?-
IN THE FIRST EQUATION? 27
THE SECOND? 32
ARE YOU SURE? YES

HERE IS A PAIR OF LINEAR EQUATIONS.
WHAT VALUES OF 'X' AND 'Y' WILL SATISFY BOTH OF THEM?
3X - 1Y = 7
2X + 4Y = 1Ø
X = 2
Y = 1

SOLVE THIS WORD PROBLEM:

A BOAT GOES UPSTREAM AT 6 MILES AN HOUR AND DOWNSTREAM AT
1Ø MILES AN HOUR. WHAT IS ITS SPEED IN STILL WATER?

SPEED OF BOAT IN STILL WATER...8

ARE YOU SURE? YES

NAME= TEACHR

TYPE OF PROBLEM	PROBLEMS TRIED	NO. RIGHT
WORD PROBLEMS	2	1
STAND. SIMUL.	2	1
LINEAR (HARD)	2	1
LINEAR (EASY)	3	1
ADD & SUB. 'X'	2	2
ARITHMETIC	4	4
NEGATIVES	2	2
INVERSES	4	4
SUBSTITUTION	1	Ø
WORK WITH 2 EQ	2	1
MUL EVERY TERM	1	1

←

3.2.2 Stringcomp and Student Programming

The foregoing examples showed that Stringcomp can accept and analyze formal mathematical expressions given as student responses in programmed lessons. In Algebra Learning Tool 2 the student is asked to write the formal sclution for X in AX+B=C; in Algebra Learning Tool 4 he attempts to set dcwn a linear equation corresponding to the verbal statement of a problem. In each case relatively simple Stringcomp programs suffice to analyze faulty student inputs for specific errors of many kinds.

Stringcomp extends the capabilities of conventional instructional languages in two other important ways:
(1) It can be programmed to simulate a mathematical or physical process that can be investigated experimentally by the student.
(2) It can be programmed by the student himself.
Stringcomp enables the student to write programs not only for carrying out numerical processes but also for carrying out formal processes and explaining their operation along the way. Writing programs of this kind should lead a student to a deeper insight into the nature of a process than working dozens of numerical examples of the usual sort.

Students have written Stringcomp programs for formal manipulation of mathematical processes of several kinds, including formal differentiation and expansion of polynomials. We illustrate with typescripts of three such programs.

The first program takes any linear equation and solves it by formal algebraic manipulation. The second program does the same thing with any pair of simultaneous linear equations in standard

form using formal substitution. The third program accepts any polynomial equation and formally solves for all prime linear factors, in the manner explained in the accompanying text. Note that these programs describe the manipulations they are performing as they perform them.

```
GIVE ME A LINEAR EQUATION IN X ...
17X - 2 +5X = 11X - 13
I AM NOW BREAKING THE EQUATION INTO INDIVIDUAL TERMS
+17X - 2 +5X = +11X - 13
NOW I'M COLLECTING TERMS
+17X +5X - 11X = +2 - 13
NOW I'M COMBINING TERMS
11X = -11
THE ANSWER IS X = -1
```

```
GIVE ME TWO EQUATIONS IN THE FORM AX+BY=C:

2X+3Y=18
3X+4Y=25

FIRST I WILL SOLVE FOR 'X' IN THE SECOND EQUATION.

I WILL DO THIS BY
    (1) SUBTRACTING 4Y FROM BOTH SIDES:
3X + 4Y - 4Y = 25 - 4Y
3X = 25 - 4Y

AND (2) DIVIDING BOTH SIDES BY 3:
3X/3 = (25 - 4Y)/3
X = (25 - 4Y)/3

THEN I SUBSTITUTE FOR 'X' IN THE FIRST EQUATION:
2((25 - 4Y)/3) + 3Y = 18

AND SIMPLIFY:
16.66667 - 2 666667Y + 3Y = 18
16.66667 + .3333333Y = 18

THEN I SOLVE FOR 'Y' BY
    (1) SUBTRACTING 16.66667 FROM BOTH SIDES:
16.66667 + .3333333Y - 16.66667 = 18 - 16.66667
.3333333Y = 18 - 16.66667

AND (2) DIVIDING BOTH SIDES BY .3333333:
.3333333Y/.3333333 = (18 - 16.66667)/.3333333

AND I GET
Y = 4

THEN I SUBSTITUTE FOR Y IN THE SECOND EQUATION:
3X + 4(4) = 25
3X + 16 = 25

AND SOLVE FOR X BY
    (1) SUBTRACTING 16 FROM BOTH SIDES:
3X + 16 - 16 = 25 - 16
3X = 9

AND (2) DIVIDING BOTH SIDES BY 3:
3X/3 = (9)/3
AND, FINALLY, I GET:
X = 3
```

FACTORING POLYNOMIALS

This program was written primarily as a demonstration of Stringcomp's string manipulation features, but could conceivably be of some value as a lesson on the standard procedure used in factoring polynomials into prime linear factors. The program accepts the equation of a polynomial to any degree and proceeds to factor it, explaining each step along the way. Notice that the program can find only prime linear factors [(X+4), (X-3), etc.]. If no factors are found, the final result will be essentially the same as the initial equation.

The problem of writing this program can be divided into two parts: the first primarily string manipulation and the second algebraic.

The function of the first part is to (1) accept the equation, (2) break the equation into individual terms, (3) collect the terms on one side of the equation, (4) combine the terms of the same power and, finally, (5) store in an array of subscripts the coefficients of each term.

With this information in hand, the second part begins. As the program explains, the only possible linear factors must be among the quotients of the factors of the first term and the factors of the last. The program first finds all the possibilities and then tries them, performing synthetic division on the polynomial each time it finds one that works. Finally the solutions are printed in factor form.

```
THIS PROGRAM DEMONSTRATES THE PROCEDURE USED IN FACTORING
POLYNOMIALS INTO PRIME LINEAR FACTORS.

GIVE ME AN EQUATION TO ANY DEGREE...X↑4 + 4X↑3 = 3X↑3 + 12X↑2

FIRST I BREAK THE POLYNOMIAL INTO INDIVIDUAL TERMS.

+X↑4 +4X↑3 = +3X↑3 +12X↑2

NEXT I COLLECT TERMS.

+1X↑4 +4X↑3 -3X↑3 -12X↑2 = ∅

THEN I COMBINE TERMS.

+1X↑4 +1X↑3 +-12X↑2 = ∅

I KNOW FROM ALGEBRA THAT THE ONLY POSSIBLE PRIME LINEAR FACTORS
ARE THE QUOTIENTS OF THE FACTORS OF THE TERM OF THE HIGHEST
POWER AND THE FACTORS OF THE LOWEST.

THE POSSIBLE FACTORS ARE:
1
2
3
4-
6
12
-1
-2
-3
-4
-6
-12

OF THESE,
3
-4
WORKED

SO WHEN I FACTOR THE POLYNOMIAL I GET:

(X-3)(X+4)(1X↑2) = ∅
←
```

The following two typescripts show how Stringcomp can be used to help a student investigate a physical process. In the first example the student merely specifies a series of physical media for a light refraction process; Stringcomp plots the resulting path of a light beam as it passes from one medium to another. In the second example the student interacts with a program, representing a missile trajectory problem, converging to the solution in a series of trials with successively improved inputs.

In these examples the student has considerable freedom to study a process. An obviously interesting extension could come if it were possible to test the student's understanding of the process studied. This might be done through giving him a problem to solve in explicit form, letting him use the simulation program as an experimental tool to help him gain insight. While he is doing this, an instructional program would try to monitor the student's work and diagnose his difficulties.

Stringcomp also has proved to be of value in this new kind of instructional application. It has been used as a programming system for embedding an instructional monitor of the kind just outlined. This work is described at length in the next section.

```
THIS PROGRAM WILL PLOT THE PATH OF A BEAM OF LIGHT AS IT PASSES
THROUGH DIFFERENT MEDIA.

HOW MANY DIFFERENT MEDIA DO YOU WANT? 6

MEDIUM...AIR    INDEX OF REFRACTION? 1
MEDIUM...GLASS    INDEX OF REFRACTION? 1.5
MEDIUM...WATER    INDEX OF REFRACTION? 1.33
MEDIUM...DIAMOND    INDEX OF REFRACTION? 2.42
MEDIUM...WATER    INDEX OF REFRACTION? 1.33
MEDIUM...AIR    INDEX OF REFRACTION? 1

WHAT IS THE ANGLE OF INCIDENCE? 45

AIR-------------------------------------------------------------
AIR              .
AIR                .
AIR                  .
GLASS-----------------------------------------------------------
GLASS                    .
GLASS                     .
GLASS                      .
WATER-----------------------------------------------------------
WATER                          .
WATER                           .
WATER                            .
DIAMOND---------------------------------------------------------
DIAMOND                             .
DIAMOND                              .
DIAMOND                               .
WATER-----------------------------------------------------------
WATER                                  .
WATER                                   .
WATER                                    .
AIR-------------------------------------------------------------
AIR                                         .
AIR                                          .
AIR                                           .
+
```

```
YOU ARE IN COMMAND OF A GROUND-TO-AIR MISSILE WHOSE JOB IS TO
BRING DOWN AN ENEMY TARGET.

AFTER YOU GIVE THE REQUIRED INFORMATION (ALL IN FEET, PLEASE)
WATCH THE PROGRESS OF THE MISSILE ON THE RADAR SCREEN.  THE
'.' REPRESENTS THE MISSILE   THE '*' IS THE SHADOW OF THE TARGET
AND '[ ]' IS THE TARGET ITSELF.

WHAT IS THE DIAMETER OF THE TARGET? 10
HOW FAR AWAY IS THE TARGET? 500
WHAT IS ITS ALTITUDE? (BETWEEN 0 AND 200) 50
WHAT IS THE VELOCITY (IN FT/SEC) OF THE MISSILE? 300
AT WHAT ANGLE IS THE MISSILE TO BE SHOT? (IN DEGREES) 45

THE MISSILE WILL REACH THE TARGET ZONE IN 2.357023 SECS
THE TIME INTERVAL ON THE RADAR SCREEN WILL BE .2357023


TIME  (0 FEET)                                          (200 FEET)
0       (          *                                          )
 .24    (        . *                                          )
 .47    (          *          .                               )
 .71    (          *                     .                    )
 .94    (          *                                .         )
1.18    (          *                                          )!
1.41    (          *                                          )!
1.65    (          *                                          )!
1.89    (          *                                          )!
2.12    (          *                                          )!
2.36    (        [ ]                                          !
SORRY, YOU MISSED BY 267.2222 FEET.


WE'RE LOADING UP A NEW MISSILE TO USE ON THE SAME TARGET.
WHAT IS THE VELOCITY (IN FT/SEC) OF THE MISSILE? 300
AT WHAT ANGLE IS THE MISSILE TO BE SHOT? (IN DEGREES) 20

THE MISSILE WILL REACH THE TARGET ZONE IN 1.77363 SECS
THE TIME INTERVAL ON THE RADAR SCREEN WILL BE .177363
```

```
TIME  (Ø FEET)                                              (2ØØ FEET)
Ø        (             *                                        )
 .18     (   .         *                                        )
 .35     (       .     *                                        )
 .53     (         .   *                                        )
 .71     (           . *                                        )
 .89     (             *  .                                     )
1.Ø6     (             *    .                                   )
1.24     (             *      .                                 )
1.42     (             *      .                                 )
1.6      (             *        .                               )
1.77     (           [ ]          .
SORRY, YOU MISSED BY 26.32Ø73 FEET.
```

WE'RE LOADING UP A NEW MISSILE TO USE ON THE SAME TARGET.
WHAT IS THE VELOCITY (IN FT/SEC) OF THE MISSILE? 3ØØ
AT WHAT ANGLE IS THE MISSILE TO BE SHOT? (IN DEGREES) 15

THE MISSILE WILL REACH THE TARGET ZONE IN 1.72546 SECS
THE TIME INTERVAL ON THE RADAR SCREEN WILL BE .172546

```
TIME  (Ø FEET)                                              (2ØØ FEET)
Ø        (             *                                        )
 .17     (   .         *                                        )
 .35     (     .       *                                        )
 .52     (       .     *                                        )
 .69     (         .   *                                        )
 .86     (           . *                                        )
1.Ø4     (             .*                                       )
1.21     (             .*                                       )
1.38     (             .*                                       )
1.55     (           . *                                        )
1.73     (         .[  ]
SORRY, YOU MISSED BY 6.296228 FEET.
```

WE'RE LOADING UP A NEW MISSILE TO USE ON THE SAME TARGET.
WHAT IS THE VELOCITY (IN FT/SEC) OF THE MISSILE? 3ØØ
AT WHAT ANGLE IS THE MISSILE TO BE SHOT? (IN DEGREES) 17

THE MISSILE WILL REACH THE TARGET ZONE IN 1.74282 SECS
THE TIME INTERVAL ON THE RADAR SCREEN WILL BE .174282

```
TIME  (Ø FEET)                                              (2ØØ FEET)
Ø        (              ¤                                      )
 .17     (   .          ¤                                      )
 .35     (       .      ¤                                      )
 .52     (           .  ¤                                      )
 .7      (              .¤                                     )
 .87     (              ¤                                      )
1.Ø5     (              ¤.                                     )
1.22     (              ¤ .                                    )
1.39     (              ¤ .                                    )
1.57     (              ¤ .                                    )
1.74     (           [  ]
SORRY, YOU MISSED BY .6678966 FEET.
```

WE'RE LOADING UP A NEW MISSILE TO USE ON THE SAME TARGET.
WHAT IS THE VELOCITY (IN FT/SEC) OF THE MISSILE? 3ØØ
AT WHAT ANGLE IS THE MISSILE TO BE SHOT? (IN DEGREES) 16.5

THE MISSILE WILL REACH THE TARGET ZONE IN 1.738248 SECS
THE TIME INTERVAL ON THE RADAR SCREEN WILL BE .1738248

```
TIME  (Ø FEET)                                              (2ØØ FEET)
Ø        (              ¤                                      )
 .17     (   .          ¤                                      )
 .35     (       .      ¤                                      )
 .52     (           .  ¤                                      )
 .7      (              .¤                                     )
 .87     (              .¤                                     )
1.Ø4     (              ¤.                                     )
1.22     (              ¤.                                     )
1.39     (              ¤.                                     )
1.56     (              ¤.                                     )
1.74     (           [ .]
```

CONGRATULATIONS! A DIRECT HIT!

←

4. Instructional Monitors

One of our major research goals has been to design and develop an "integrated" instructional system of wide applicability. Our purpose is to combine the principal advantages of traditional, computer-directed teaching strategies which derive from a programmed instruction background with the richness and flexibility available in a system having a good student programming or simulation language.

In traditional CAI the computer maintains control of the interaction sequence by severely constraining the possible responses of the student at any given time. Typically the framework of a particular teaching program is (or could be) displayed as a (finite) network of information- and question-presentation blocks, followed by calls for response and decision blocks. Some flexibility is available in the path a student takes through the network but the basic style casts the student in the role of a decision-maker of very limited scope. Some of the Stanford work of Suppes and Smallwood present good examples of this type of approach carried as far as it seems sensible to go. It is, for example, particularly well-suited to arithmetic problems in addition and subtraction, where the student responses are reasonably constrained by the nature of the questions.

Student programming languages, such as Stringcomp and Logo (described in Section 5), provide the student with a very flexible and rich capability to express procedures for solving problems. Special-purpose programming languages for simulation, such as the ENPORT language for dynamic physical systems study*,

*Karnopp, D.C., and Rosenberg, R.C., *Analysis and Simulation of Multiport Systems.* Cambridge, Mass.: M.I.T. Press, (1968).

offer similar capabilities constrained by the particular operations permitted to the student. Typical forms of problems using such languages include:
"Find a procedure that will..."
"For what values of the system parameters will the level just overflow?"
"Write a program to predict the impact point, given the initial velocity."

The major difficulty with the use of programming languages in an instructional context in CAI is the inability of current teaching systems to monitor unanticipated solutions made by the student.

Another aspect of the contrast between the two approaches is the information required from the (human) instructor in defining a problem or problem sequence. In traditional CAI a huge amount of very detailed information is required, including correct and plausible incorrect responses at interaction points. Sometimes it seems as if the instructor must incorporate every error ever liable to be made, as well as some unlikely ones. In a system based on a student-programming language facility, a problem statement and a few clues to be parceled out typically serve to define a problem. The low level of information required partly is dictated by the inability of the teaching system to use more data. (The usual notions of correct and incorrect responses and of teaching strategies do not lead to particularly useful information for judging freely constructed procedures.)

We have for some time been interested in the possibilities of an instructional system that could combine the best of both worlds, i.e.,

(1) maintain some measure of control over the interaction sequence, directing the student toward a solution to the problem; and
(2) permit a considerable amount of flexibility to the student in creating, testing, and judging his own solutions to the problem.

This part of the report has two parts, the first a brief recapitulation of the chain of developments leading us to an integrated instructional monitor design, and the second a description of SIMON (SImple MONitor), a prototype monitor operating at BBN.

4.1 A Brief History of Monitor Development

The first efforts toward monitor development involved a simulation language called ENPORT (op. cit.). ENPORT simulates the dynamic behavior of a broad class of physical systems of different types, among which are electrical circuits, mechanical systems of particles and rigid bodies, hydraulic and fluid systems, and others. One reason this language was chosen is that, although a large variety of physical systems can be considered, the description of systems is accomplished by using a simple, succinct graphical notation called bond graphs. It was hoped that a monitor could be designed that would be able to interpret and critique any (bond graph) model a student might propose as a solution to a problem. Further, it was hoped that a close connection could be made between observed dynamic behavior of a model and essential characteristics of a bond graph, thus enabling a monitor to comment helpfully to a student about the implications of observed data in relation to a proposed model.

Both of these hopes were overly optimistic, in that the fundamental relations between bond graphs and dynamic response are not so closely coupled that a monitor can be given the requisite interpretive abilities for teaching. Attempts were made to limit the class of allowable bond graph models in order to simplify the interpretation problem, but they were unsuccessful.

The experience with ENPORT, in which the simulation language remained the key and ways to get a monitor to interpret it were sought, did not deter us from trying the same pattern again on a modified language called WORDPORT. Elements were defined in common English terms, such as SPRING, PIPE, TANK, and SWITCH, and a minimum set of attributes was defined (e.g., SWITCH could be ON or OFF). The intent was to create a language rich enough to permit interesting configurations to be realized (such as a WORDPORT model of a washing machine), but still allow successful monitoring of any model and its possible behaviors. Unfortunately, as the WORDPORT system was enhanced to permit models with interesting behavior to be made, the difficulties of interpreting the models from a controlled instructional viewpoint increased severely. Once again we were forced to terminate the approach.

Finally we transformed our view so that, instead of trying to limit a language to permit complete monitor control, we accepted a rich language to begin with (STRINGCOMP) and started developing monitor control capabilities systematically as we were able, along with input and output conventions and incorporating various programs necessary toward building an instructional system. The result to date is a design known as SIMON, discussed next.

Both of these hopes were overly optimistic, in that the fundamental relations between bond graphs and dynamic response are not so closely coupled that a monitor can be given the requisite interpretive abilities for teaching. Attempts were made to limit the class of allowable bond graph models in order to simplify the interpretation problem, but they were unsuccessful.

The experience with ENPORT, in which the simulation language remained the key and ways to get a monitor to interpret it were sought, did not deter us from trying the same pattern again on a modified language called WORDPORT. Elements were defined in common English terms, such as SPRING, PIPE, TANK, and SWITCH, and a minimum set of attributes was defined (e.g., SWITCH could be ON or OFF). The intent was to create a language rich enough to permit interesting configurations to be realized (such as a WORDPORT model of a washing machine), but still allow successful monitoring of any model and its possible behaviors. Unfortunately, as the WORDPORT system was enhanced to permit models with interesting behavior to be made, the difficulties of interpreting the models from a controlled instructional viewpoint increased severely. Once again we were forced to terminate the approach.

Finally we transformed our view so that, instead of trying to limit a language to permit complete monitor control, we accepted a rich language to begin with (STRINGCOMP) and started developing monitor control capabilities systematically as we were able, along with input and output conventions and incorporating various programs necessary toward building an instructional system. The result to date is a design known as SIMON, discussed next.

when the student is satisfied with his solution, he asks the monitor to check it;

the interaction continues in this style, with the monitor parceling out comments and help until either the student's program is correct or no further help can be given by SIMON. (As a last resort in the latter case, the monitor's solution may be listed.)

Figure 1 is a schematic diagram showing the major components of the instructional system, and the principal flows of information. Observe that the monitor maintains control over all access to various parts of the system. For example, the student cannot examine the behavior of the monitor's solution until he has made a commitment to a set of variables for the problem.

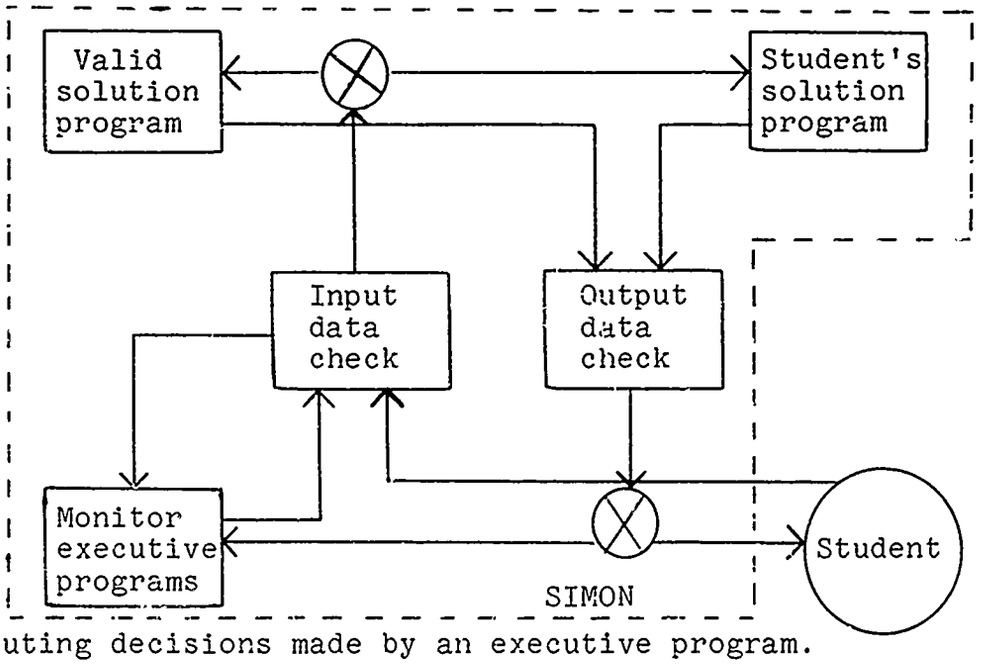


Figure 1. Diagram of SIMON showing information flow.

The figure shows that either SIMON or the student can initiate data for a problem trial, route it to either the monitor's program or the student's program, and direct the results to either the monitor or the student. This flexibility permits the student to say, for example, "You (SIMON) choose some trial data, apply it to my program, and tell me the results." Or it enables the monitor to comment, "The trial data you have chosen are not sensible because ... ."

Some of the requests the student can make of SIMON are (the list is not complete):

RUN YOU ... student asks monitor to run its solution program.
RUN ME ... student asks monitor to run his program.
COMPOSE ... student informs monitor that he wishes to write a program to solve the problem.
CHECK ME ... student asks monitor to check his program.
HELP ... student asks monitor for any help it can give him when his program fails the monitor's checking procedure.
WHY ... student asks monitor to list the trial values used and results obtained in its last checking procedure.
SELECT ... student selects input and output variables from monitor's comprehensive list of legal names.

A detailed description of SIMON is given in Appendix 3.1.

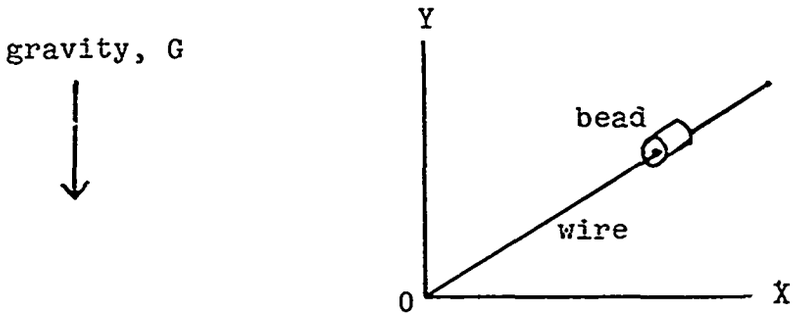
4.3 An Illustrative Application of the SIMON Monitor

Figure 2 shows a SIMON problem statement of a sliding bead problem for a hypothetical student. The student is to develop a String-comp program that expresses a valid solution of the problem.

Bead on a Wire

A bead with a hole in it is free to slide on a piece of straight wire. If the wire is fixed in some position relative to the gravity field, how long will it take the bead to reach the end of the wire?

In order to discuss this problem, let us use the coordinates and names shown in the figure below.



Put the bead at some initial position (XØ, YØ), set the gravitational constant (G), and you can find the time (T) it takes for the bead to reach point 0. The mass of the bead is M.

When you think you can predict the time, write a program to do so and ask SIMON to check it for you.

Figure 2. Problem statement for a bead on a wire.

A sample transaction is shown in Fig. 3, in which student typing is either preceded by an asterisk, or underscored. First, the student must select the variables (from a list in this case) that he thinks are relevant for this problem. Then he chooses input values and observes the results (i.e., the three "RUN YOU" requests in Fig. 3a).

He then feels confident enough (see Fig. 3b) to write a program of his own ("COMPOSE"), which he uses once ("RUN ME"). The result is correct for that case. After that he asks to be checked by SIMON ("CHECK ME"), and his program produces erroneous results on the first set of trial values, at which point he asks "WHY" and seeks "HELP". Now a major error is exposed by the information that M (the mass) is not relevant to this problem.

In Fig. 3c we see that the student changes his declaration of variables by eliminating M, modifies his program, and repeats the process. This time he tests his program on known values and results, is satisfied, and gets checked out successfully by SIMON. Notice in passing that SIMON no longer asks for a value of M in operating either the student's program or its own solution.

BEAD ON A WIRE

(NOTE. XØ AND YØ MUST BE GREATER THAN OR EQUAL TO ZERO.
EXPRESS INFINITY AS 1Ø↑36)

```
*SELECT

SHALL I LIST THE LEGAL VARIABLE NAMES?  YES
XØ       INITIAL POSITION ON X AXIS
YØ       INITIAL POSITION ON Y AXIS
M        MASS OF BEAD
G        GRAVITATIONAL ACCELERATION
T        TIME
HOW MANY INPUT VALUES DO YOU THINK YOU WILL NEED?  4
LIST THEM
1 XØ
2 YØ
3 M
4 G

HOW MANY OUTPUT VARIABLE NAMES?  1
LIST IT
1 T
*RUN YOU

XØ= 5
YØ= 5
M= 3
G= 32

          T=       .79Ø5694
*RUN YOU

XØ= [5]    = 1Ø
YØ= [5]    = 1Ø
M= [3]     = 3
G= [32]    = 32

          T=      1.118Ø34
*RUN YOU

XØ= [1Ø]   = 5
YØ= [1Ø]   = 1Ø
M= [3]     = 3
G= [32]    = 32

          T=       .8838835
```

Figure 3a.

```
*COMPOSE
GO AHEAD
*1.1 THETA=ATN(XØ,YØ)
*1.2 S=(YØ↑2+XØ↑2)↑.5
*1.3 ACC=G/M
*1.4 T=(2*S/ACC)↑.5 IF YØ<>Ø
*1.5 T=1Ø↑36 IF YØ=Ø
*1.6 TYPE T
```

Comments on user program.

1.1 Theta=Arctan(XØ/YØ)

1.2 $S=\sqrt{(YØ)^2+(XØ)^2}$

1.3 ACC=G/M

1.4 $T=\sqrt{2 \cdot S/ACC}$ if YØ≠Ø

1.5 $T=10^{36}$ if YØ=Ø

1.6 Type value of T

```
*RUN ME

XØ= [5]     = 5
YØ= [1Ø]    = Ø
M= [3]      = 3
G= [32]     = 32

YOUR PROGRAM---
            T=      1*1Ø↑36

*CHECK ME

MY PROGRAM---
            T=      .3535534
YOUR PROGRAM---
            T=      .5149418
YOU'RE A LOSER

*WHY

HERE ARE THE VALUES I GOT FROM YOUR PROGRAM
XØ=1
YØ=1
G=32
M=3
MY RESULTS:          YOUR RESULTS:
T=.3535534           .5149418
*HELP
YOU HAVE SELECTED MORE VALUES THAN YOU WILL NEED TO SOLVE THIS
PROBLEM.  YOU DO NOT NEED:  M
```

Figure 3b.

```
*SELECT

SHALL I LIST THE LEGAL VARIABLE NAMES?  NO
HOW MANY INPUT VALUES DO YOU THINK YOU WILL NEED?  3
LIST THEM
1 XØ
2 YØ
3 GRAV
GRAV ISN'T ON THE LIST
DO YOU WANT TO SEE THE LIST  NO
3 G

HOW MANY OUTPUT VARIABLE NAMES?  1
LIST IT
1 T
*TYPE STEP 1.3

1.3 ACC=G/M
*1.3 ACC='G*SIN(THETA)'

*RUN ME

XØ= 1
YØ= 1
G= 32

YOUR PROGRAM---
            T=      .3535534

*CHECK ME

MY PROGRAM---
            T=      .3535534
YOUR PROGRAM---
            T=      .3535534

MY PROGRAM---
            T=      .559Ø17
YOUR PROGRAM---
            T=      .559Ø17

MY PROGRAM---
            T=      1*1Ø↑36
YOUR PROGRAM---
            T=      1*1Ø↑36
YOU'RE A WINNAH
```

Figure 3c.

Now that we have seen how SIMON operates from the student's point of view we might well ask, "What must the instructor do to set up the bead problem?" This is the task of problem definition, and it consists of four main parts --

a problem description (text information);

information about legal variable names, and variables relevant to the problem at hand;

a valid solution program; and

information about checking the student's program (e.g., trial values, number of trials).

The first step after the problem description is to establish an appropriate master list of variables, and to designate the relevant input and output variables. If the problems are of a related type (e.g., particle motion problems), a comprehensive master list may already exist containing variables like T (time), G (gravitational acceleration), M (mass), and XV (x-velocity). The instructor may add to the master list, edit it, or start anew. In the example of the bead we have chosen to start afresh.

→DO PART 92

WOULD YOU LIKE TO SEE THE CURRENT NAME LIST? NO

WILL YOU USE THIS LIST? NO

HOW MANY ENTRIES IN LIST OF POSSIBLE NAMES? 5

LIST NAME AND THEN COMMENT PLEASE

1 XØ INITIAL POSITION ON X AXIS
2 YØ INITIAL POSITION ON Y AXIS
3 M MASS OF BEAD
4 G GRAVITATIONAL ACCELERATION
5 T TIME

All input and output variables must be taken from this list.

Next the input and output variables are defined. When the student selects his input variables they will be compared to this list, making it evident where he has overspecified and/or underspecified quantities in the problem.

LIST THE INPUT NAMES REQUIRED FOR THIS PROBLEM
1 XØ
2 YØ
3 G
4

LIST THE OUTPUT NAMES REQUIRED FOR THIS PROBLEM
1 T
2

Next the instructor decides how many test cases constitutes a reasonable check on the validity of the student's solution program, and what sets of trial values are important and informative (to both the student and SIMON). The "CHECK" request causes these trials to be run according to the checking rules given below.

```
HOW MANY TRIES? 5

TRY NO. 1
XØ=1
YØ=1
G=32

TRY NO. 2
XØ=Ø
YØ=5
G=32

TRY NO. 3
XØ=5
YØ=Ø
G=32

TRY NO. 4
XØ=-45
YØ=3Ø
G=32

TRY NO. 5
XØ=577
YØ=466
G=3.?

SHOULD I RE-TRY THE VALUE WHICH LOST FIRST? YES

HOW MANY TIMES SHOULD I TRY FOR CHECKING? 3

WHAT PERCENT ERROR IS ACCEPTABLE? 1
```

Starting with "try one" SIMON will concede that the program is correct when it does three trial sets correctly. If an error is found, the next check will start with that try on which the error occurred. The result (T) in this case must be accurate to within one percent.

Finally a brief segment of text is accepted, followed by a solution program. The rules for writing the solution programs are those of the STRINGCOMP language.

```
TYPE IN YOUR INSTRUCTIONS, END EACH LINE WITH ALTMODE
TYPE  DONE  WHEN FINISHED

*BEAD ON A WIRE
*(IT WILL BE LEFT TO YOU TO DETERMINE THETA IF YOU FEEL YOU
*NEED IT.  YØ MUST BE >=Ø.  EXPRESS INFINITY AS 1Ø↑36)
*
*DONE

THE EXAMPLE GOES IN PART 1Ø.

→1Ø.1 THETA=ATN(XØ,YØ),S=SQRT(YØ↑2+XØ↑2),ACC='G*SIN(THETA)'
→1Ø.2 T=SQRT(2*S/ACC) IF YØ<>Ø
→1Ø.3 T=1Ø↑36 IF YØ=Ø
→1Ø.4 TYPE T
→TYPE PART 1Ø

1Ø.ØØ DONE IF I1 IS "RUN ME" ;MY PROGRAM
1Ø.1 THETA=ATN(XØ,YØ),S=SQRT(YØ↑2+XØ↑2),ACC='G*SIN(THETA)'
1Ø.2 T=SQRT(2*S/ACC) IF YØ<>Ø
1Ø.3 T=1Ø↑36 IF YØ=Ø
1Ø.4 TYPE T
→
```

The basic information for this problem is summarized in Fig. 4. 'VLIST' is the master list, 'VNAM' is the relevant input set, 'ONAM' is the output set, and the 'TRY' array has the trial values. For the flexibility of interaction possible in this problem, not very much data is required.

```
←LOAD PHIL-BEAD ON A WIRE
←TYPE PART 1Ø
1Ø.ØØ DONE IF I1 IS "RUN ME" ;MY PROGRAM
1Ø.Ø4 PRINT "YØ MUST BE GREATER THAN OR EQUAL TO Ø",# IF YØ<Ø
1Ø.Ø5 PRINT "XØ MUST BE GREATER THAN OR EQUAL TO Ø",# IF XØ<Ø
1Ø.Ø6 DONE IF XØ<Ø @ YØ<Ø
1Ø.1 THETA-ATN(XØ,YØ),S=SQRT(YØ↑2+XØ↑2),ACC='G*SIN(THETA)'
1Ø.2 T=SQRT(2*S/ACC) IF YØ:>Ø
1Ø.25 T=Ø IF YØ=Ø & XØ=Ø
1Ø.3 T=1Ø↑36 IF YØ=Ø
1Ø.4 TYPE T

←TYPE VLIST[I] FOR I=1:1:VLIST[Ø]
    VLIST[1]= XØ
    VLIST[2]= YØ
    VLIST[3]= M
    VLIST[4]= G
    VLIST[5]= T

←TYPE VNAM[I] FOR I=1:1:VNAM[Ø]
    VNAM[1]= XØ
    VNAM[2]= YØ
    VNAM[3]= G

←TYPE ONAM[I] FOR I=1:1:ONAM[Ø]
    ONAM[1]= T

 TYPE TRY[I,J] FOR J=1:1:VNAM[Ø] FOR I=1:1:TRY[Ø,Ø]
    TRY[1,1]=       1
    TRY[1,2]=       1
    TRY[1,3]=      32

    TRY[2,1]=       Ø
    TRY[2,2]=       5
    TRY[2,3]=      32

    TRY[3,1]=       5
    TRY[3,2]=       Ø
    TRY[3,3]=      32

    TRY[4,1]=     -45
    TRY[4,2]=      3Ø
    TRY[4,3]=      32

    TRY[5,1]=     577
    TRY[5,2]=     466
    TRY[5,3]=      32
```

Figure 4. Summary of information required for the bead problem.

4.4 Use of SIMON for a Complex Problem

One of the characteristics of real problems, as opposed to purely academic ones, is that frequently they require extensive organization of a number of simple fragments of knowledge. The problem discussed in this section is intended to illustrate the use of SIMON in presenting a complex situation. The problem is complex because a number of component parts must be fitted together, even though the most complicated physical law needed is the one describing motion of a particle under constant acceleration.

The problem statement indicates that there are three parts to be solved, each one being more general than the part before. The transcripts cover the first two parts, and it should be possible to extrapolate from them in further development of the problem. The student, in addition to piecing together a straight line trajectory (for the helicopter) and a parabolic trajectory (for the package upon release), must bear in mind constraints such as "...the helicopter must never approach closer to the ground than 200 feet ...". Such a problem adds up to a challenge to one's ability to organize a solution from known fragments. As subsequent complications are introduced, the student's understanding of how to extend the previous solution to get the maximum value from it is enhanced.

Description of the Rescue Problem

In this problem, which has several parts you are the pilot of a helicopter which is to drop a package near to an injured mountain climber. The eventual goal is for you to predict the time of release of a package, given a trajectory for the helicopter and

information about wind conditions, so that the package lands as near to the injured man as you can get it.

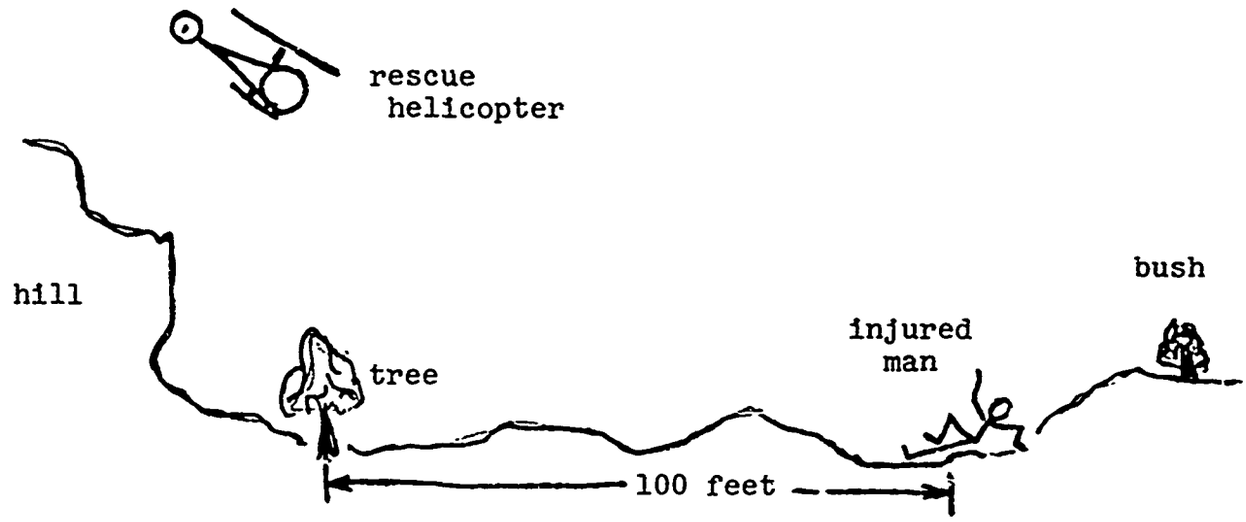
To begin the investigation we will assume that there is no significant wind, and allow you to choose any helicopter trajectory you wish. When you are familiar with the behavior of the package under a variety of conditions, you may write a program to predict the landing point of the package.

When your program predicts successfully, the wind will arise dramatically, thereby introducing additional loading accelerations on the package as it falls. Again, when you feel comfortable with the behavior of the package, write a program to predict the point of impact.

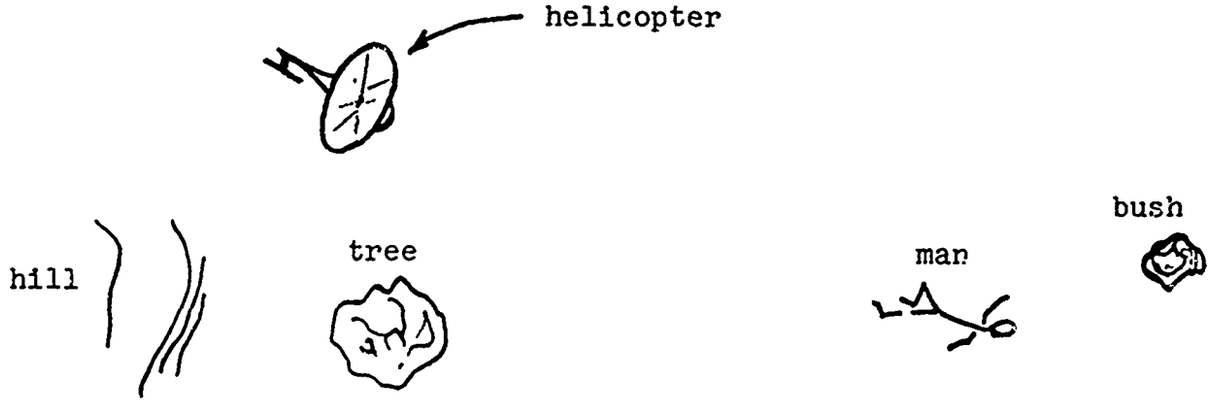
Finally, we shall approach the objective, which is to determine the best time to release the package (i.e., the one which makes the point of impact closest to the man), given a particular helicopter trajectory and wind conditions.

Examine Fig. 5a in which two views of the rescue situation are presented. The injured man lies about 100 feet from a tree, and we can establish a sight line between the two objects. In Fig. 5b a set of coordinates are displayed which will allow us to discuss the problem in convenient terms. "X" has its origin at the tree, and extends in the man's direction. "Y" originates at the tree, and is perpendicular to "X". "Z" also originates at the tree, and is perpendicular to "X" and "Y" (and hence the ground). Thus the position of the tree is (0,0,0), and that of the man is (100,0,0).

The trajectory of the helicopter is always a straight line which may be described by a position and a velocity at time zero. Between time zero and the time of package release the helicopter must never approach closer to the ground than 200 feet, for safety's sake.



(a) A side view of the rescue problem.



(b) A top view of the rescue problem.

Figure 5a. Two views of the rescue problem.

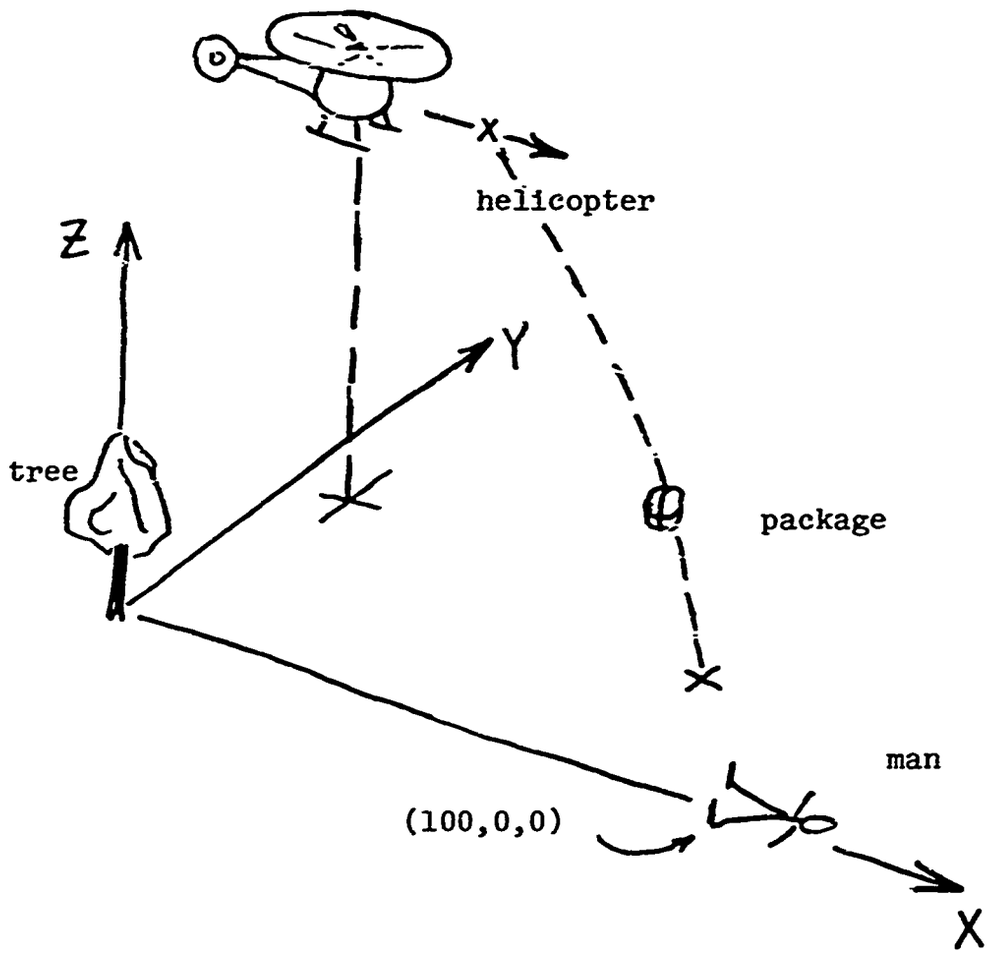


Figure 5b. Coordinates for the rescue problem.

Discussion of the Rescue Problem Transactions

Figure 6 (parts a–h) present a representative interaction between a hypothetical student and SIMON in the solution of part 1 of the rescue problem. First a summary of the rules and requests for interacting with SIMON are given, followed by a discussion of relevant and legal variables (see Fig. 6b). In subsequent trials

the input list given by the student controls the interaction (whether correct or not).

The student, after selecting variables, tries a few runs to see how the package behaves under various conditions (Figs. 6c, 6d, and 6e). When he is fairly confident that he understands how the package is behaving, he writes his own program and operates it (see Fig. 6f). Wisely, the first values he chooses to test it are ones for which he already knows the correct results. (He might have done even better to use simpler conditions.)

Unfortunately, not only does his program predict the point of impact and time incorrectly but also he omitted an important result (DIS). He is informed of this omission immediately, but puzzles over sources of error, and asks for help. One comment is that the mass of the package is not important in the current situation. It is left to the student to verify this information, and to figure out why it is true.

The student edits the variable list and edits his program (see Fig. 6g). Then he tries his program again on a known case (see Fig. 6h) and it predicts correctly. Encouraged by this turn of events, he asks SIMON to check his program, which is done. The result is some brief reinforcement (a verbal pat on the back) and a harder problem.

RESCUE PROBLEM PART 1 (NO WIND)

TO WRITE A PROGRAM, TYPE STEPS INTO SIMON JUST AS YOU WOULD INTO STRINGCOMP. YOUR PROGRAM MUST BEGIN IN PART 1 AND MAY USE PARTS 1 THROUGH 9. THERE IS A SECTION IN SIMON WHICH WILL ALLOW YOU TO SPECIFY WHICH VARIABLES YOU THINK YOU WILL NEED FOR YOUR PROGRAM AND ANOTHER SECTION WHICH WILL ALLOW YOU TO ASSIGN VALUES TO THESE VARIABLES. THUS, NO DEMAND OR ACCEPT COMMANDS WILL BE NECESSARY IN YOUR PROGRAM. USE THE OUTPUT NAMES SPECIFIED IN THE PROBLEM INSTRUCTIONS. YOU MAY TYPE YOUR ENTIRE PROGRAM WITH THE USE OF THE 'TYPE ME' COMMAND, OR YOU MAY SPECIFY A SINGLE STEP OR PART TO BE TYPED BY SAYING 'TYPE' FOLLOWED BY THE SPECIFIC STEP OR PART. THE 'ERASE' COMMAND WORKS THE SAME WAY. SIMON WILL ALLOW YOU TO EXECUTE ANY DIRECT COMMAND BY TYPING 'EXECUTE' FOLLOWED BY THE SPECIFIC COMMAND TO BE EXECUTED. SHOULD YOU BREAK OUT OF SIMON CONTINUE BY TYPING 'DO PART RETURN'.

TO CHOOSE VARIABLE NAMES TYPE	SELECT
TO RUN MY PROGRAM TYPE	RUN YOU
TO RUN YOUR PROGRAM TYPE	RUN ME
TO WRITE A PROGRAM TYPE	COMPOSE
TO CHECK YOUR PROGRAM AGAINST MINE TYPE	CHECK ME
TO SEE WHAT SIMON TRIED TYPE	WHY
IF YOU GET STUCK TYPE	HELP ME
TO GIVE VALUES TO THE SELECTED VARIABLES TYPE	VALUES
TO SEE ALL OR ANY SECTION OF YOUR PROGRAM TYPE	TYPE --
TO DELETE ALL OR ANY SECTION OR YOUR PROGRAM TYPE	ERASE --
TO EXECUTE A STRCOMP COMMAND TYPE	EXECUTE --
FOR A LIST OF LEGAL COMMANDS TYPE	LIST

Figure 6a. Rescue problem - Part 1.

```
*RUN YOU
YOU MUST SELECT THE VARIABLE NAMES YOU WISH TO USE BEFORE YOU MAY
RUN A PROGRAM.
USE THE 'SELECT' COMMAND.

*SELECT

SHALL I LIST THE LEGAL VARIABLE NAMES?  YES
XH0     X COORDINATE OF HELICOPTER AT TIME 0
YH0     Y      "        "      "       "    "
ZH0     Z      "        "      "       "    "
VXH     (CONSTANT) X VELOCITY COMPONENT OF HELICOPTER
VYH         "      Y    "          "       "       "
VZH         "      Z    "          "       "       "
TR      TIME PACKAGE IS RELEASED
AZP     (CONSTANT) Z ACCELERATION OF PACKAGE  I.E. GRAVITY
TL      TIME PACKAGE LANDS
XP      X COORDINATE OF PACKAGE
YP      Y      "      "      "
ZP      Z      "      "      "
DIS     DISTANCE BETWEEN MAN AND PACKAGE AT TL
MP      MASS OF PACKAGE

HOW MANY INPUT VALUES DO YOU THINK YOU WILL NEED?  9
LIST THEM
1 XH0
2 VH0
VH0  ISN'T ON THE LIST
DO YOU WANT TO SEE THE LIST  NO
2 YH0
3 ZH0
4 VXH
5 VYH
6 VZH
7 TR
8 AZP
9 MP

HOW MANY OUTPUT VARIABLE NAMES?  5
LIST THEM
1 TL
2 XP
3 ZP
4 YP
5 DIS
```

Figure 6b.

```
*RUN YOU

XH0= 0
YH0= 0
ZH0= 0
VXH= 0
VYH= 0
VZH= 0
TR= 0
AZP= -32
MP= 0

DO YOU WANT TO SEE THE TRAJECTORIES PLOTTED? YES

HELICOPTER TOO LOW; TROUBLE.

*RUN YOU

XH0= [0]    = 0
YH0= [0]    = 0
ZH0= [0]    = 200
VXH= [0]    = 0
VYH= [0]    = 0
VZH= [0]    = 0
TR= [0]     = 0
AZP= [-32]  = -32
MP= [0]     = 0

DO YOU WANT TO SEE THE TRAJECTORIES PLOTTED? YES
          TL=       3.535534
          XP=       0
          YP=       0
          ZP=       0
         DIS=     100

X POSITION  0 FEET                                          200 FEET
    0          (                                              ^
    0          (                                              ^
    0          (                                              ^
    0          (                                              ^
    0          (                                            . ^
    0          (                                          .   ^
    0          (                                        .     ^
    0          (                                      .       ^
    0          (                                    .         ^
    0          (                                  .           ^
    0          (                                .             ^
    0          (                              .               ^
    0          (                            .                 ^
    0          (                          .                   ^
    0          (                        .                     ^
    0          (                      .                       ^
    0          (                    .                         ^
    0          (                  .                           ^
    0          (                .                             ^
    0          (              .                               ^
    0          (            .                                 ^
    0          (          .                                   ^
    0          (        .                                     ^
    0          (      .                                       ^
    0          (    .                                         ^
    0          (  .                                           ^
    0         !(                                              ^
```

Figure 6c.

```
* RUN YOU

XH0= [0]    = 0
YH0= [0]    = 0
ZH0= [200]  = 200
VXH= [0]    = 20
VYH= [0]    = 0
VZH= [0]    = -10
TR= [0]     = 0
AZP= [-32]  = -32
MP= [0]     = 10

DO YOU WANT TO SEE THE TRAJECTORIES PLOTTED? YES
          TL=      3.236818
          XP=     64.73635
          YP=      0
          ZP=      0
         DIS=     35.26365

X POSITION  0 FEET                                          206 FEET
  -12.94727   (                                                 ^
   -6.473635  (                                               ^ )
    0         (                                             ^  )
    6.473635  (                                           .^   )
   12.94727   (                                        .  ^    )
   19.42091   (                                     .     ^    )
   25.89454   (                                  .        ^    )
   32.36818   (                               .           ^    )
   38.84181   (                            .              ^    )
   45.31545   (                         .                 ^    )
   51.78908   (                      .                    ^    )
   58.26272   (                   .                       ^    )
   64.73635   (                                           ^    )
   71.20999   !(                                          ^    )
```

Figure 6d.

```
* RUN YOU

XH0= [0]    = -10
YH0= [0]    = 0
ZH0= [200]  = 200
VXH= [20]   = 20
VYH= [0]    = 10
VZH= [-10]  = 20
TR= [0]     = 5
AZP= [-32]  = -32
MP= [10]    = 10


DO YOU WANT TO SEE THE TRAJECTORIES PLOTTED? YES
          TL=     10
          XP=    190
          YP=    100
          ZP=     -1.144409*10↑-5
         DIS=    134.5362
```

Figure 6e.

```
*COMPOSE

GO AHEAD

*1.1 RT=(-VZH-SQRT(VX**ZH**2-2*AZP*(ZH0+VZH*TR)))/AZP

*1.2 TL=TR+RT/2*MP**********(RT

*1.2 TL=TR+RT/(2*MP)

*1.3 XP=XH0+VZH**XH*TL

*1.4 YP=YH0+VYH*TL

*1.5 ZPP**=0

*1.6 TYPE TL,XP,YP,ZP

*RUN ME

XH0= [-10] = -10
YH0= [0]   = 0
ZH0= [200] = 200
VXH= [20]  = 20
VYH= [10]  = 10
VZH= [20]  = 20
TR= [5]    = 5
AZP= [-32] = -32
MP= [10]   = 10


YOUR PROGRAM---
          TL=      5.25
          XP=     95
          YP=     52.5
          ZP=      0

 I DON'T SEE A 'DIS' IN YOUR PROGRAM

*HELP

YOU HAVE SELECTED MORE VALUES THAN YOU WILL NEED TO SOLVE THIS PROBLEM
YOU DO NOT NEED:
MP
```

Figure 6f.

```
*SELECT

SHALL I LIST THE LEGAL VARIABLE NAMES?  NO
HOW MANY INPUT VALUES DO YOU THINK YOU WILL NEED?  8
LIST THEM
1 XH0
2 YH0
3 ZH0
4 VXH
5 VYH
6 VZH
7 TR
8 AZP

HOW MANY OUTPUT VARIABLE NAMES?  5
LIST THEM
1 TL
2 XP
3 YP
4 ZP
5 DIS


*COM# TYPE ME

1.0; STUDENT PROGRAM LOCATION
1.1 RT=(-VZH-SQRT(VZH↑2-2*AZP*(ZH0+VZH*TR)))/AZP
1.2 TL=TR+RT/(2*MP)
1.3 XP=XH0+VXH*TL
1.4 YP=YH0+VYH*TL
1.5 ZP=0
1.6 TYPE TL,XP,YP,ZP

*COMPOSE

GO AHEAD

*1.2 TL**=TR+RT

*1.55 DIS=SQRT((100-XP)↑2+YP↑2)

*1.6 TYPE TL,XP,YP,ZP,DIS
```

Figure 6g.

```
*RUN ME

XH0= -10
YH0= 0
ZH0= 200
VXH= 20
VYH= 10
VZH= 20
TR= 5
AZP= -32


YOUR PROGRAM---
          TL=      10
          XP=     190
          YP=     100
          ZP=       0
         DIS=     134.5362
*CHECK ME


MY PROGRAM---
          TL=      14.24193
          XP=     -15.1614
          YP=      48.48386
          ZP=       0
         DIS=     124.9513
YOUR PROGRAM---
          TL=      14.24193
          XP=     -15.1614
          YP=      48.48386
          ZP=       0
         DIS=     124.9513

MY PROGRAM---
          TL=      18.80414
          XP=     100.0829
          YP=     362.3917
          ZP=       0
         DIS=     362.3917
YOUR PROGRAM---
          TL=      18.80414
          XP=     100.0829
          YP=     362.3917
          ZP=       0
         DIS=     362.3917
YOU'RE A WINNAH

LOADING A NEW PROBLEM
```

Figure 6h.

In Fig. 7 (a–f) the problem has been broadened to allow accelerations in all three coordinate directions. The style of interaction is similar to that of part 1 and culminates in a successful student program after two minor corrections (see Figs. 7d and 7e).

In a more realistic treatment of the problem situation we could introduce a wind velocity and some loading factors, thus turning the problem into one involving true dynamics. In fact, the student could be led to deduce the wind loading characteristics for himself by conducting a series of experiments in which he drops the package and sees where it lands.

The information required by SIMON of the instructor in setting up parts 1 and 2 is summarized in Appendix 3.2. It includes the legal and relevant variable lists, the solution programs, and the trial values for checking.

```
DROPIT PART 2
(THERE IS NOW A WIND FACTOR TO BE TAKEN INTO ACCOUNT)


*SELECT

SHALL I LIST THE LEGAL VARIABLE NAMES? YES
XH0     X COORDINATE OF HELICOPTER AT TIME 0
YH0     Y     "       "       "       "  "
ZH0     Z     "       "       "       "  "
VXH     (CONSTANT) X VELOCITY COMPONENT OF HELICOPTER
VYH         "      Y    "          "       "      "
VZH         "      Z    "          "       "      "
MP      MASS OF PACKAGE
TR      TIME PACKAGE IS RELEASED
AXP     (CONSTANT) X ACCELERATION OF PACKAGE
AYP         "      Y       "       "      "
AZP         "      Z       "       "      "
TL      TIME PACKAGE LANDS
XP      X COORDINATE OF PACKAGE
YP      Y     "       "    "
ZP      Z     "       "    "
DIS     DISTANCE BETWEEN MAN AND PACKAGE AT TL

HOW MANY INPUT VALUES DO YOU THINK YOU WILL NEED? 10
LIST THEM
1 XH0
2 YH0
3 ZH0
4 VXH
5 VYH
6 VZH
7 TR
8 AXP
9 AYP
10 AZP

HOW MANY OUTPUT VARIABLE NAMES? 5
LIST THEM
1 TL
2 XP
3 YP
4 ZP
5 DIS
```

Figure 7a. Rescue problem - Part 2.

```
*RUN YOU

XH0= -100
YH0= -100
ZH0= 500
VXH= 20
VYH= 20
VZH= 20
TR= 10
AXP= -5
AYP= -5
AZP= -332**2


DO YOU WANT TO SEE THE TRAJECTORIES PLOTTED? YES
          TL=     17.26884
          XP=     113.2867
          YP=     113.2867
          ZP=        0
         DIS=     111.0632

X POSITION  0 FEET                                                  859 FEET
   70.92463   (                                                        )
   85.46232   (                                                        )
  100         (                                                        )
  113.2168    (                                                        )
  123.7918    (                                                        )
  131.7249    (                                                        )
  137.0163    (                                                        )
  139.6659    (                                                        )
  139.6736    (                                                        )
  137.0396    (                                                        )
  131.7638    (                                                        )
  123.8461    (                                                        )
  113.2867    (                                                        )
  100.0854   !(                                                        )
```

Figure 7b.

```
* RUN YOU

XH0= [-100]= -100
YH0= [-100]= -100
ZH0= [500] = 500
VXH= [20]  = 20
VYH= [20]  = 20
VZH= [20]  = 20
TR= [10]   = 10
AXP= [-5]  = 0
AYP= [-5]  = 0
AZP= [-32] = -32


DO YOU WANT TO SEE THE TRAJECTORIES PLOTTED? YES
          TL=     17.26884
          XP=    245.3768
          YP=    245.3768
          ZP=       0
         DIS=    285.2091

X POSITION  0 FEET                                          859 FEET
   70.92463   (                                                 )
   85.46232   (                                                 )
  100         (                                                 )
  114.5377    (                                                 )
  129.0754    (                                                 )
  143.613     (                                                 )
  158.1507    (                                                 )
  172.6884    (                                                 )
  187.2261    (                                                 )
  201.7638    (                                                 )
  216.3015    (                                                 )
  230.8391    (                                                 )
  245.3768    (                                                 )
  259.9145   !(                                                 )
```

Figure 7c.

```
*COMPOSE

GO AHEAD

*1.1 RT=(-VZH-(SQRT(VZH↑2-2*AZP*(VZH*TR+ZH0))))/AZP

*TTY# TYPE ME

1.0; STUDENT PROGRAM LOCATION
1.1 RT=(-VZH-(SQRT(VZH↑2-2*AZP*(VZH*TR+ZH0))))/AZP

*1.2 TL=TR+RT

*1.3 XP=XH0+AXP*RT↑2/2

*1.4 YP=YH0+VYH*TL+AYP*RT↑2/2

*1.5 ZP=0

*1.6 DIS=SQRT((100-XP)↑2+YP↑2

*1.6 DIS=((100-XP)↑2+YP↑2)↑/+---.5

*1.7 TYPE TL,XP,YP,ZP,DIS

*RUN ME

XH0= [-100]= -100
YH0= [-100]= -100
ZH0= [500] = 500
VXH= [20]  = 20
VYH= [20]  = 20
VZH= [20]  = 20
TR= [10]   = 10
AXP= [0]   = 0
AYP= [0]   = 0
AZP= [-32] = -32


YOUR PROGRAM---
          TL=      17.26884
          XP=    -100
          YP=     245.3768
          ZP=        0
         DIS=     316.5593


*1.3 XP=XH0+AZP*RT↑2/2 +VYH*TL
```

Figure 7d.

```
* RUN ME

XH0= -100]= -100
YH0= [-100]= -100
ZH0= [500] = 500
VXH= [20]  = 20
VYH= [20]  = 20
VZH= [20]  = 20
TR= [10]   = 10
AXP= [0]   = 0
AYP= [0]   = 0
AZP= [-32] = -32


YOUR PROGRAM---
          TL=     17.26884
          XP=    -600
          YP=     245.3768
          ZP=       0
         DIS=     741.7613

*1.3 XP=XH0+VXH*TL+AXP*RT↑2/2

* RUN ME

XH0= [-100]= -100
YH0= [-100]= -100
ZH0= [500] = 500
VXH= [20]  = 20
VYH= [20]  = 20
VZH= [20]  = 20
TR= [10]   = 10
AXP= [0]   = 0
AYP= [0]   = 0
AZP= [-32] = -32


YOUR PROGRAM---
          TL=     17.26884
          XP=     245.3768
          YP=     245.3768
          ZP=       0
         DIS=     285.2091
```

Figure 7e.

```
*CHECK ME

MY PROGRAM---
          TL=     13.75
          XP=    -48.4375
          YP=    -41.40625
          ZP=     -7.629395*10↑-6
         DIS=    154.1044
YOUR PROGRAM---
          TL=     13.75
          XP=    -48.4375
          YP=    -41.40625
          ZP=      0
         DIS=    154.1044

MY PROGRAM---
          TL=     13.75
          XP=   -239.0625
          YP=   -246.0938
          ZP=     -7.629395*10↑-6
         DIS=    418.9577
YOUR PROGRAM---
          TL=     13.75
          XP=   -239.0625
          YP=   -246.0937
          ZP=      0
         DIS=    418.9576

YOU'RE A WINNAH
```

Figure 7f.

5. Programming and Problem-Solving: The Logo Programming Language

This part of the research is a preliminary exploration of two hypotheses: first, that through teaching the use of an appropriate programming language, it is possible to impart the fundamental concepts and skills of formal thinking and problem-solving; second, that these concepts and skills can be taught to elementary school children.

Our first argument for teaching programming as a core subject is like the classical justification often given for teaching Latin. Many educators insist on the value of teaching this language while avowing complete indifference as to the value of actually knowing it. The teaching of Latin is said to foster an analytic way of thinking, a sensitivity to formal aspects of language, an insight into features of English, discipline of thought, and many other intellectual virtues. One might be justly skeptical about whether the teaching of Latin achieves these ends. Recent experiences point to programming as a more likely subject. The exploratory work of the past few years clearly suggests that the teaching of programming can have important side effects such as instilling in pupils the notions and skills of logical thinking and expression.

For example, the use of computers and programming languages helps students strive for self-consciousness and literacy about *the process of solving problems*. High school students can seldom *say anything* about how they worked toward the solution of a problem. They lack the habit of discussing such things and they lack the vocabulary necessary to do so. Using a programming language remedies both of these deficiencies. Further, the study

of programming helps pupils acquire very early the habit of organizing their approach to mathematical problems. It is highly plausible that this training will better prepare them to develop systematic habits of thought in the more murky areas of problem-solving they will have to meet later, in school and elsewhere.

To investigate these hypotheses, we designed a new programming language, called Logo, as the conceptual and operational framework for a mathematics laboratory curriculum. We taught the language to a summer class of seventh grade children at the Hanscom School in Lincoln, Massachusetts, in July 1967. The ensuing discussion is based upon our teaching experiment.

The basic requirements set for Logo were as follows.

(1) Third graders should be able to use it for simple tasks with very little preparation.

(2) Its structure should embody mathematically important concepts with minimal interference from programming conventions.

(3) It should permit the expression of mathematically rich nonnumerical algorithms as well as numerical ones.

A good typical example of a nonnumerical problem suitable for young children is translating English into "Pig Latin". There are many problems of this sort which children already know and like. The student thinks at first that he understands such problems perfectly because, with a little prodding, he can give a loose verbal description of his procedure. But he finds it

difficult to make this description precise partly for lack of formal habits and partly for lack of a suitably expressive language. The initial value of using Logo becomes apparent when the student attempts to make the computer perform his procedure. At this point the process of transforming loose verbal descriptions into precise formal ones becomes possible and, in this context, seems natural and enjoyable to children.

Logo Operations

In Logo one manipulates three kinds of objects, called *letters*, *words*, and *sentences*. A Logo letter can be an alphabetic character, a numerical character, or one of several special characters such as *, %, @. A Logo word is any sequence of letters not separated by a space. Thus, a Logo word can be an English word such as DOG, a nonsense word such as SHRDLU, or a number word such as 2471. A Logo sentence is any sequence of words separated by spaces. Thus, DOG SHRDLU 2471 can be a Logo sentence, as can an English sentence such as THE SKY IS BLUE.

Several types of operations can be performed on Logo letters, words, and sentences. These include *combining*, *printing*, *counting*, *extracting*, and *naming*, illustrated by the following examples. (Quotation marks are used to delimit the objects on which the operations are performed as well as the objects that result.)

Examples of the combining operations are:
WORD OF "CAT" AND "DOG" = "CATDOG"
SENTENCE OF "CAT" AND "DOG" = "CAT DOG"

To perform these operations on the computer terminal (which is an ordinary teletype console and looks to the children like a typewriter) one uses the PRINT operation. Thus, if one types PRINT WORD OF "SUN" AND "DAY" and presses the carriage return key to indicate the end of input, Logo advances the paper one line and then types back the word SUNDAY.

Examples of the extracting operations are:

```
FIRST OF "CAT" = "C"
BUTFIRST OF "CAT" = "AT"
LAST OF "CAT" = "T"
BUTLAST OF "CAT" = "CA"
FIRST OF "THE FAT CAT" = "THE"
BUTLAST OF "THE FAT CAT" = "THE FAT"
```

Logo operations can be compounded. Thus,

```
WORD OF FIRST OF "CAT" AND LAST OF "CAT" = "CT"
FIRST OF LAST OF BUTLAST OF SENTENCE OF "HERE" AND "WE ARE" = "W"
```

At a very early stage we introduce the concept of *naming*. Any Logo object can be used to name any other Logo object. Thus "SHRDLU" can be the name of the word "DOG" and "MOONBEAM" the name of the sentence "STARRY NIGHT 5". To name an object, the user types NAME. Logo then asks him, first, to type in the object (the "thing") to be named and, then, to type in the name to be given to that object. We illustrate using the examples just mentioned. The user's inputs are underscored to distinguish them from Logo's outputs.

```
NAME
THING:  "DOG"
NAME OF THING:  "SHRDLU"
```

NAME
THING: "STARRY NIGHT 5"
NAME OF THING: "MOONBEAM"

When we want to use a Logo object as a literal (i.e., as itself), we enclose it between quotation marks. Thus, PRINT FIRST OF "SHRDLU" causes Logo to type S. But we sometimes want to use an object as a name for something else. To do this, we enclose the object between slashes.* Thus, with the naming relations just defined, PRINT /SHRDLU/ causes Logo to type DOG, and PRINT BUTFIRST OF /MOONBEAM/ causes Logo to type NIGHT 5.

Up to this point we have written no programs but we have introduced and discussed a number of important mathematical ideas:

(1) that we can operate upon objects of different kinds and that we must distinguish carefully between them. This principle does not have to be formulated; the children learn it themselves since Logo prints error comments, for example, when asked to form a word from two sentences or from a single word;

(2) that operations can be classified according to the kinds and numbers of objects they take as arguments and the kinds of objects they give as values;

(3) that naming a thing and finding a named object are well-defined formal operations (these become as natural and concrete as adding or subtracting numbers);

* This use of quotation marks and slashes is not introduced to make subtle philosophical distinctions. Straightforward and very practical problems create a need for the distinction between "DOG" and /DOG/ in the context of Logo.

(4) that *operations* can be compounded, inverted, and generally manipulated as objects in their own right.

Programs

We introduce the concept of procedure (or program) in Logo through examples.

Suppose we wish to create a new operation in Logo which takes a word and makes a new word from it by "doubling" the first, i.e., "CAT" becomes "CATCAT". We first choose a name for the new operation--say DOUBLE. The effect we would like to get is shown by the following input and its associated output.

```
PRINT DOUBLE OF "CAT"        CATCAT
```

To get this effect we first inform Logo that we are about to define a procedure called DOUBLE. We then tell Logo how to DOUBLE a word. The procedure is written below. Our instructions to Logo are listed under the column labeled Input; Logo's responses (and our parenthesized comments) are listed under the column labeled Effect or Output.

Input	Effect or Output
TO DOUBLE /W/	(Logo understands that we are about to define a procedure and that it is to store the next lines of input as the definition, or the recipe, for DOUBLING. It also understands that the name "W" will be used within the procedure for the word to be DOUBLED.)
RETURN WORD OF /W/ AND /W/	(RETURN is the Logo term for "the answer is".)
END	(END marks the end of the definition. Logo confirms by typing the following message.) DOUBLE DEFINED
PRINT DOUBLE OF "CAT"	CATCAT
PRINT DOUBLE OF "HO"	HOHO

It is instructive to children to consider several procedures that are equivalent in that they have the same effect.

```
TO DOUBLE /TROUBLE/
RETURN WORD OF /TROUBLE/ AND /TROUBLE/
END
```

```
TO DOUBLE /W/
NAME
THING:  WORD OF /W/ AND /W/
NAME OF THING:  "NEW"
RETURN /NEW/
END
```

Once a procedure is defined, it can be used in the same way as the built-in Logo procedures (operations) such as WORD and FIRST, including their use in defining new procedures. Thus, for example, we can write PRINT WORD OF DOUBLE OF SECOND OF "YO HO" AND "HA" and Logo will respond HOHOHA.

The next example shows a nonterminating recursive procedure. The instruction DO, followed by the name of a procedure, simply invokes Logo to perform that procedure.

```
TO MUMBLE /JUMBLE/
PRINT /JUMBLE/
DO MUMBLE OF /JUMBLE/
END
```

Thus, PRINT MUMBLE OF "GO" causes Logo to type GO
GO
GO
.
.
.

and so on, ad infinitum if the program is not stopped externally.

The recursion in this program (i.e., the fact that it calls on itself) did not seem to pose the slightest difficulty for the children--on the contrary, they found this idea quite natural! They also liked the idea that the program goes on forever. To allow the construction of recursive procedures which will eventually stop, we need a *conditional operation*. In Logo this has the explicit form of a question.

IS /X/ /Y/

This question is followed by one (or both) of the instructions IFYES ---, IFNO ---. The dashes stand for some Logo operation or procedure, such as STOP, or RETURN /X/, or DO MUMBLE OF /JUMBLE/. The effect is self-explanatory. The operation following the IFYES will be performed only if /X/ is /Y/, i.e., "X" and "Y" name the same value. The following example shows the use of a conditional operation, and introduces the use of "the empty word" (i.e., the word with no letters, known to Logo by the name

"EMPTY"). The procedure was written by one of the children. At this point they were inventing and programming their own procedures.

Input	Effect or Output
TO TRIANGLE /WORD/	("WORD" is the name of the word we are dealing with)
IS /WORD/ /EMPTY/	("EMPTY" is the name of the empty word)
IFYES STOP	(If /WORD/ is not empty, the program will not stop; it will proceed to the next instructions in the sequence)
PRINT /WORD/	
DO TRIANGLE OF BUTFIRST OF /WORD/	
END	TRIANGLE DEFINED
DO TRIANGLE OF "TIME"	TIME IME ME E

The children took very easily to writing recursive procedures like this one. These provide a particularly good way of exercising the use of variable names: in each "round" of the procedure TRIANGLE, the name "WORD" has a different value, i.e., BUTFIRST of its previous value.

Recursion

The study of recursive procedures in Logo provides a valuable approach to understanding the ideas underlying mathematical reasoning. These ideas appear difficult because they are so

unlike anything else the student has met in his formal mathematical work. A good illustration is provided by the conceptual difficulty surrounding the use of "mathematical induction". This fundamental and exciting mathematical principle is in fact extremely simple and ought to be presented very early. In Logo this principle is embedded in a more general class of recursive principles. These can be systematically investigated in a range of cases of increasing difficulty starting from the trivial recursion in our procedure MUMBLE and proceeding through examples such as the following.

We want to define a recursive procedure called REVERSE which takes a word and reverses the letters in it, i.e., writes it backwards. Thus,

```
REVERSE OF "CAT" = "TAC"
```

We observe in class (as an instance of a method used very often) that a possible plan for the construction of a procedure is to reduce the problem of reversing a long word to that of reversing shorter words. Thus, suppose we wish to reverse "ELEPHANT". Let us pretend that a magician is available to reverse any shorter word, e.g., "LEPHANT". Then we could use his services by asking him to reverse this and then putting "E" on the end:

```
REVERSE OF "ELEPHANT" = "TNAHPELE"
                      = WORD OF "TNAHPEL" AND "E"
                      = WORD OF REVERSE OF "LEPHANT" AND "E".
```

Let us try to describe this for all words:

```
REVERSE OF /W/ = WORD OF REVERSE OF BUTFIRST OF /W/
                 AND  FIRST OF /W/
```

Once this is understood it is easy to see that we can dispense with the magician by applying the same process until we get down to a single letter word, which can be reversed trivially (by doing nothing).

Thus we have the procedure -

```
TO REVERSE /W/
IS /W/ FIRST OF /W/
IFYES RETURN /W/
IFNO RETURN WORD OF REVERSE OF BUTFIRST OF /W/ AND FIRST OF /W/
END
```

Heuristics

Experience in writing Logo procedures is equally valuable in teaching the heuristic aspects of mathematical work. One instance is provided by the frequently used technique of *chaining*, i.e., composing procedures from other procedures. We proposed to the children the project of making a procedure to translate English sentences into Pig Latin sentences. Pig Latin was defined in the following simplified form:

If the English word begins with a consonant, the Pig Latin word is obtained by transferring the consonant to the end and adding "AY", i.e., by the procedure:

```
TO PIG1 /W/
NAME
THING:  WORD OF BUTFIRST OF /W/ AND FIRST OF /W/
NAME OF THING:  "IT"
RETURN WORD OF /IT/ AND "AY"
END
```

Thus, PIG1 OF "CAT" is "ATCAY".

If the word begins with a vowel, the procedure is

```
TO PIG2 /W/
RETURN WORD OF /W/ AND "WAY"
END
```

Thus, PIG2 OF "IF" is "IFWAY".

Our problem is to define an operation, let's call it PIGIFY, to turn English sentences into Pig Latin sentences. We do this by using the heuristic procedure of dividing the problem into easier subproblems and *worrying separately* about how to do them and how to put the results together.

Our successive simplifications are:

Operation PIGIFY on sentences uses Operation PIG on words.
Operation PIG on any word uses

> Operation PIG1 on words beginning with consonants, and
> Operation PIG2 on words beginning with vowels.

The heuristic approach is further illustrated by the observation that when we are defining PIG we can forget about how PIG1 and PIG2 work--as long as we know what they do. To define PIG we need a test for whether a letter is a vowel. So let's *pretend* we have one. What would it do? It could have many forms, one of which is

VOWEL OF /LETTER/ = "TRUE" if /LETTER/ is a vowel.
VOWEL OF /LETTER/ = "FALSE" if /LETTER/ is not a vowel.

Using this form, we could define PIG as follows.

```
TO PIG /W/
IS VOWEL OF FIRST OF /W/ "TRUE"
IFYES RETURN PIG2 OF /W/
IFNO RETURN PIG1 OF /W/
END
```

Thus, the problem of defining PIG is reduced to that of defining VOWEL. We define VOWEL by means of a yet simpler operation, called CONTAINS, which will operate on a letter and a word and have the value "TRUE" if the letter is contained in the word. (Note that the procedure CONTAINS actually reduces the problem to the yet simpler problem of deciding whether /L/ is the first letter of the word.)

```
TO CONTAINS /L/ AND /W/
IS /W/ /EMPTY/
IFYES RETURN "FALSE"              (RETURN causes the procedure to
IS /L/ FIRST OF /W/               terminate at this point by return-
IFYES RETURN "TRUE"               ing the answer "FALSE")
DO CONTAINS OF /L/ AND BUTFIRST OF /W/
END
```

Now, at last,

```
TO VOWEL /LETTER/
RETURN CONTAINS OF /LETTER/ AND "AEIOU"
END
```

All this might seem sophisticated and confusing when read in five minutes. But, when approached gradually through classroom discussion and, especially, with the opportunity to try things out on the terminal, the ideas become vividly clear. Twelve-year-old pupils understood these ideas and were able to use them in other problems such as "translating" English into cryptic codes of their own invention.

The experiment illustrates the use of programming languages in teaching mathematical thinking and problem-solving. This way of using computers appears to have rich implications for the theory and practice of education, though a great deal of work is needed to develop and evaluate it further.

6. Summary

We have investigated several lines of approach to the use of computers in education, including programmed teaching, instructional monitoring, and programming. The Mentor, Stringcomp, Simon, and Logo programming systems were designed and used as an integral part of these investigations. We have described their capabilities and demonstrated their uses in many examples, and have thereby shown that systems like these can have interesting, and non-trivial, instructional applications.

Specific technical innovations have included:

(1) The use of conditional statements to express student/computer interactions in programmed teaching discourses.

(2) The use of string manipulation to express formal non-numerical relationships in programmed teaching and testing.

(3) The use of programming to express problem-solving procedures which can be performed and tested by a programmed monitor.

(4) The use of programming as a means of teaching mathematical thinking and the skills of clear and precise expression.

The educational contributions made possible by these developments have been suggested through specific examples in medicine, mathematics, physics, and other subjects. Further developments along some of these same lines promise to yield important educational benefits. For example, using capabilities already shown in the Logo teaching experiment, the use of programming ideas and experience will make possible a new kind of mathematics curriculum. In this curriculum, the teaching of appropriate programming languages such as Logo will serve as the conceptual and operational framework for the teaching of mathematics. The experiment has

indicated that exceptional motivational and cognitive advantages can derive from this approach.

New directions for extending some of the techniques explored presage the development of teaching programs with greatly increased interpretive capabilities. For example, instructional monitors might be developed that can make reasonable diagnoses of student learning difficulties. The Simon monitor, as described above, enables a student to express his own solution procedure for an assigned problem and then tries to check whether the procedure is valid by applying it to known cases. It permits a student to express his own solution procedure and, in most practical applications, has a rather good expectation of determining that a procedure is not valid. But, in case a procedure is not valid, Simon is unable to determine why.

To build a monitor with true diagnostic capabilities is not a trivial task. But we do know, in principle, how to make a program follow the steps of a student who is not constrained by a stereotyped pattern and how to diagnose his difficulties on the way. To do this requires that we give the program some knowledge of how to uncover and recognize learning difficulties and, further, provide a means of generating and testing hypotheses about the difficulties. No such experiments can be done without a great deal of work, but certain skills in subjects as diverse as music, physics, languages, and navigation appear to lend themselves to such treatment.

Appendix 1.1 Description of the Mentor Programming Language

Mentor is a specialized programming language designed expressly for the creation of instructional dialogues between a student and a computer. Instructional material created in it may range from the familiar and simple question-and-answer "programmed instruction" to logically complex tutorial dialogues. The language imposes few artificial barriers to the creation of instructional interactions even when used for describing material of the more complex sort. Mentor language syntax and semantics are described and illustrated in this introductory note.

Mentor Programs

Educational material is prepared in Mentor in the form of a program. The program contains the information to be conveyed to the student and computer instructions which govern its presentation. Mentor interactions are carried out through teletypewriter exchanges between student and computer. The student types a question or answer and the machine responds in the same fashion. The material may have the form of drill exercises, multiple-choice quizzes, "branching" lessons (as in classical programmed instruction), or freer and more complex tutorial dialogues, including so-called Socratic dialogues.

An entire Mentor program is called a *text*. A text is subdivided into *contexts*, each comprising a collection of *conditional statements*. A conditional statement expresses the actions to be taken by the computer if the student types a particular input and if specified conditions describing the occurrence of previous inputs and statements are satisfied. In the simplest instance, a

statement merely associates a student's input with the computer's output. For example, in a physics lesson, such a statement might assert that, in a given context, the input "electron density" is to cause the computer to output the message "Right."

The text and each of its contexts are assigned names. We illustrate with an unserious text, called QUIZ, with contexts named ANIMALS, BASEBALL, CHESS, and TALLY whose content could be written in Mentor along the following lines.

```
TEXT QUIZ.

START ANIMALS.

CONTEXT ANIMALS.
BEGIN →  "DO YOU LIKE ANIMALS?"
"NO"  →  "THEN YOU CAN'T BE ALTOGETHER BAD.", START BASEBALL.
"YES" →  "HOW MANY HORNS DOES A UNICORN HAVE?".
"ONE" OR "1" →  "THAT'S WHAT I USED TO THINK UNTIL I LEARNED
         THAT THE UNICORN IS A MYTHICAL ANIMAL.",
         START BASEBALL.
"ZERO" OR "Ø" OR "NONE" → "I HOPE IT GROWS BACK SOON.",
         START BASEBALL.
ELSE  →  "YOU BETTER FIND OUT MORE ABOUT ANIMALS.", START
         BASEBALL.
END ANIMALS.

CONTEXT BASEBALL.
BEGIN →  "WHAT IS THE MOST POPULAR AMERICAN SPECTATOR SPORT?".
"BASEBALL" → "SORRY, YOU'RE BEHIND THE TIMES.   TV WATCHING
         IS FIRST.", START CHESS.
"TELEVISION" OR "TV" → "WATCHING TV IS NO SPORT.", START CHESS.
"GIRLS" OR "GIRL WATCHING" → "ONLY FOR BOYS.", START CHESS.
ELSE  →  "YOU BETTER STAY HOME MORE.", START TALLY.
END BASEBALL.

CONTEXT CHESS.
BEGIN → "LET'S TALK ABOUT CHESS.  HAVE YOU HEARD ABOUT MAELZEL?"
"YES" → "YOU KNOW A LOT ABOUT CHESS.", START TALLY.
ELSE  → "POOR MAELZEL WANTED TO PUT A MAN INTO A MACHINE.  WHAT
        A HUMANITARIAN!", START TALLY.
END CHESS.
```

```
CONTEXT TALLY.
BEGIN → "YOU DID SPLENDIDLY ON THIS QUIZ.  YOU GOT ZERO RIGHT
        OUT OF ZERO QUESTIONS.  BYE."
END TALLY.

END QUIZ.
```

This is how the quiz might appear to a student. (Student = S; Computer = C.)

```
C:   DO YOU LIKE ANIMALS?
S:   YES
C:   HOW MANY HORNS DOES A UNICORN HAVE?
S:   1
C:   THAT'S WHAT I USED TO THINK UNTIL I LEARNED THAT THE UNICORN
     IS A MYTHICAL ANIMAL.
     WHAT IS THE MOST POPULAR AMERICAN SPECTATOR SPORT?
S:   PROTEST RALLIES
C:   YOU BETTER STAY HOME MORE.  LET'S TALK ABOUT CHESS.  HAVE
     YOU HEARD ABOUT MAELZEL?
S:   WHAT ABOUT HIM?
C:   POOR MAELZEL WANTED TO PUT A MAN INTO A MACHINE.  WHAT A
     HUMANITARIAN!
     YOU DID SPLENDIDLY ON THIS QUIZ.  YOU GOT ZERO RIGHT OUT
     OF ZERO QUESTIONS.  BYE.
```

The structure of this quiz may be pictured by the following figure.

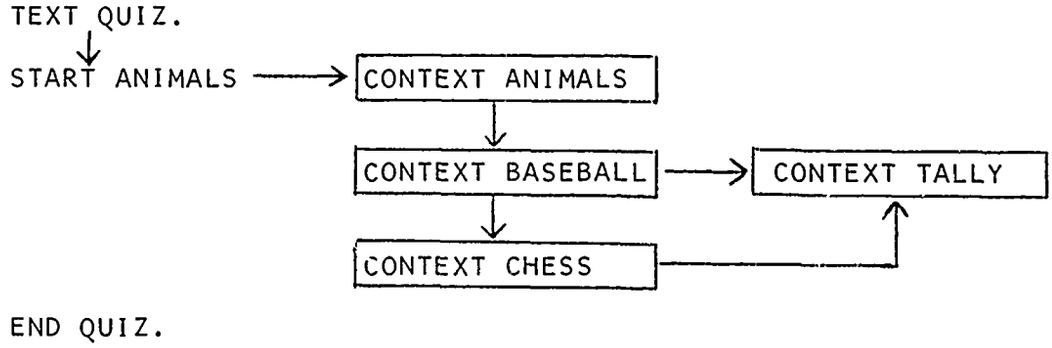


Figure 1.

The TEXT declaration establishes a name for the program. This convenience keeps programs separated within the computer and, since the name is made up by the programmer, it can also suggest the contents of the program. The name used may be made up from any combination of letters, numerals, and special characters such as *, -, $, ?, without any intervening spaces. The TEXT declaration is the first line in every program and the END declaration followed by the program name must be the last line.

Referring to Fig. 1, you can see that the text is composed of several contexts and a START instruction. As might be guessed, the START command directs the computer to enter a specific context, in this case the first one, ANIMALS. In any Mentor program, this command designates the starting context and appears in the first line after the TEXT declaration. At any point during the interaction, the program is operating in precisely one of the four contexts; we say the program is "in" a given context at any time. Generally, the program changes contexts throughout an interaction. Figure 1 shows the paths along which the program can change context in the quiz. The context names, ANIMALS, BASEBALL, CHESS, and TALLY, were made up by the programmer and their composition follows the rules for text names.

Each CONTEXT block is composed of conditional statements. Each statement is broken into two parts, a left and right side, and these are connected by an arrow. Before investigating the construction of these statements, let's examine the meaning of them in the context BASEBALL. First we look at only the material to the left of the arrows, e.g.,

```
CONTEXT BASEBALL.
BEGIN →
"BASEBALL" →
"TELEVISION" OR "TV" →
"GIRLS" OR "GIRL WATCHING" →
ELSE →
END BASEBALL.
```

When the computer enters the context BASEBALL, the BEGIN command tells it to immediately type out "WHAT IS THE MOST POPULAR AMERICAN SPECTATOR SPORT?". The machine then waits for a student response. The expected responses, each of which will be treated separately, are "BASEBALL", "TELEVISION", "TV", "GIRLS", and "GIRL WATCHING". The OR connecting some of the inputs is a logical modifier which indicates that those inputs are synonymous.

If the student types in anything not on the list, his response will activate the ELSE command and cause the typeout "YOU BETTER STAY HOME MORE." and the program will enter the context TALLY, as directed by the START command on the right side of the statement. If an expected response *is* typed by the student, the associated message will be typed to him and the program will enter the context CHESS, as directed by the START command on the right side of the statement. Notice that the expected input "YES" appears in context ANIMALS and in context CHESS. Because the machine can be "in" only <u>one</u> context at a time, each of the two "YES" answers is unambiguously related to the corresponding question in each context.

Basic Conditional Statements

As you have seen, the conditional statement is a device for describing how the actions of the computer are conditional on

the inputs of the student. The conditional part of the statement is expressed on the left side of the arrow and the action to be taken by the computer is on the right side. The left side may be one of the commands BEGIN or ELSE or it may be a piece of quoted text which is to be matched against a student input. When a match occurs, the actions designated on the right side will be carried out. In simple statements, the right side may tell the machine to type out a message (surrounding a message with quotes does this), execute a command, or both. After the actions are performed, control is returned to the student and the program waits for him to type an input.

The BEGIN command is used when the programmer wants the computer to type out a message at the beginning of a context. Referring to the Quiz once again, notice each context (except TALLY) consists of a question for the student expressed in a BEGIN statement, followed by a series of statements expressing the student's expected answers and the computer's associated actions.

Every context must have one and only one ELSE statement, for a quite simple reason. If the student types an unanticipated word or phrase, it is necessary for the machine to respond in a meaningful way to sustain the dialogue. All unanticipated responses are captured by the ELSE statement. For example, misspelled inputs (which are not recognized by a phonetic spelling encoding program incorporated in the Mentor System) will activate the ELSE statement. The right side of an ELSE statement often has the form of a cliché such as "Sorry, I didn't understand that. Please try again." If the programmer forgets to include an ELSE statement in any context, the computer will automatically insert one which has the message "I didn't understand." on the right side.

Compound Conditional Statements

With basic conditional statements and the commands TEXT, CONTEXT, END, BEGIN, ELSE, and START, it is possible to create a variety of structurally simple instructional interactions. However, more complex interactions between student and machine such as those occurring in Socratic dialogues require more programming tools. In such dialogues the machine must handle logically complex situations which depend not only on the student's immediate input but also on the history of his activity. To this end, logical variables may be created and used within Mentor programs. These variables are symbols which have either a TRUE or FALSE value. The value of a variable may be set to TRUE or FALSE in various ways during the operation of the program. For example, a logical variable can be created to record whether the student types any given input. If he does, the program will set the value of this variable to TRUE. At any time, the truth value of this variable can be used as a condition within a conditional statement. Before discussing how logical variables are defined and used in compound conditional statements, we briefly consider the general form of such statements.

The general form of the Mentor conditional statement is compactly expressed as follows:
A *conditional statement* is a *key condition* followed by an arrow (→) followed by a *subconditional statement*.
A key condition is either a quoted string denoting student input or a BEGIN command or an ELSE command or a list of quoted strings separated by ORs (synonyms) or a quantified list (discussed below).

A subconditional statement is either an *action list* (as in the basic conditional statement) or a *conditional expression*.. A conditional expression is either a *basic conditional expression* or a *composite conditional expression*. A basic conditional expression has the form:

$$[c_1 \rightarrow a_1;\ c_2 \rightarrow a_2;\ \ldots;\ c_{n-1} \rightarrow a_{n-1};\ \text{Else} \rightarrow a_n].$$

where the c_i are a set of mutually exclusive conditions (predicates defined by logical variables) and the a_i are associated actions.

A composite conditional expression has the form:

$$[c_1 \rightarrow b_1;\ c_2 \rightarrow b_2;\ \ldots;\ c_{n-1} \rightarrow b_{n-1};\ \text{Else} \rightarrow b_n]$$

where the c_i are, again, a set of mutually exclusive conditions, and the b_i are basic conditional expressions as defined above. This informal description of Mentor conditional expressions shows how they are recursively defined. A variety of concrete examples will be given in the sections that follow. We first discuss the logical variables used to express conditions in Mentor statements.

Logical Variables

A standard use of logical variables in Mentor is to indicate that a particular input has been typed by the student. This kind of logical variable is created simply by enclosing a name in parentheses immediately after the associated quoted input. Thus,

```
"ABDOMINAL EXAM" (A) → ...
"HEART EXAM" (B) → ...
```

"ABDOMINAL EXAM" and "HEART EXAM" are key conditions with A and B, respectively, denoting the associated logical variables. These parenthesized variables can be referred to in other conditional statements to determine if the associated inputs, which precede them, have been typed in. In the above example, if the student types ABDOMINAL EXAM, the variable A is set to TRUE; if he types HEART EXAM, the variable B is set to TRUE. (All variables were previously set to FALSE, at the beginning of the interaction.) Once a variable is set to TRUE, it remains unchanged until it is explicitly reset.

Let us consider how these variables A and B might be used in a conditional statement, in the following example.

```
"X-RAYS" (XR) → [A → "O.K., X-RAYS WILL BE TAKEN.",
     START TEST;
     B → "YOU SHOULD FINISH YOUR ABDOMINAL EXAMINATION
         FIRST.";
     ELSE → "YOU HAVEN'T EVEN LOOKED AT THE PATIENT'S
         HEART YET!"].
```

In this example, when the student types in X-RAYS, the machine sets XR to TRUE and begins to check the list of statements within the brackets. If A is TRUE, it types out "O.K. ..." and starts CONTEXT TEST. If B is TRUE, it types out "YOU SHOULD FINISH ...". If neither A nor B is TRUE, it checks ELSE, a special variable that is always TRUE, and types "YOU HAVEN'T EVEN ...".

The list of statements within the brackets is called a *subconditional statement*. In principle, the list may be indefinitely long. Each element of the list is an ordinary conditional statement except that the left side consists only of a logical variable. The computer tests each variable in the list until a

TRUE-valued one is found. When this occurs, the right side is evaluated and the associated actions are performed. The computer does not test beyond the first one found to be TRUE. Every list must have at least one variable which is TRUE. ELSE and INAPPROPRIATE are two special variables which are always TRUE and therefore one of them must be the last variable on the list. INAPPROPRIATE has the same effect as ELSE but, unlike ELSE, sets the logical variable associated with the key condition to FALSE. Thus, in the example, if the list after X-RAYS ended with an INAPPROPRIATE instead of an ELSE, and A and B were both FALSE, the XR variable would be reset to FALSE.

We have considered parenthesized logical variables which tell whether or not certain inputs were typed by the student. Other kinds of logical variables may be created by the functions DEFINE, LET, and RECORD. For example, the programmer may wish to know if any one or all of a particular set of inputs was typed by the student. A function called DEFINE enables him to define such a variable. This is how it might look.

```
DEFINE GLOB = "input1", "input2", "input3".
```

GLOB, the defined variable, is FALSE until any one of the student inputs in the list is typed in. Two special logical quantifiers may accompany variables defined in this way, ANY and ALL.

When GLOB is used in a subconditional, for example, it can appear only as,

```
→[ANY GLOB→
  ELSE→        ].
```

or,

```
→[ALL GLOB→
  ELSE→        ].
```

As well as being used in subconditionals, ANY and ALL can be used as key conditions in the left part of any conditional statement.

LET is a logical function, similar to DEFINE, permitting a single logical variable to take on the truth value of several others. For example,

```
LET STUFF = L, M, AVG, Q1.
```

As a result of this statement, STUFF is FALSE until ANY one or ALL of the logical variables, L, M, AVG, and Q1 is TRUE. The special modifiers ANY and ALL which were used for DEFINE variables also modify LET variables.

RECORD is a logical function which defines a variable and sets it to TRUE. It is used not with inputs but with outputs, i.e., in an action list. For example,

```
"YES" (Y) → [M→ "————————";
             N→ "————————", RECORD WOW;
             P→ "————————", RECORD YP;
             ELSE→                    ].
```

If the student types YES and N is TRUE, an output message is typed and WOW is set to TRUE. Variables are used in this way to indicate whether or not a particular branch of a conditional statement has been executed and, if so, to leave a trace. This device is of special importance in recording interaction history for subsequent interrogation.

ERASE is a logical function which may be used to set any variable to FALSE. Like RECORD, ERASE is an action. For example,

```
"HEART" (H) → [M→ "PLEASE FINISH THE EXAM.",
               ERASE HENRY;
               L→ ...;
               ELSE→ ... .].
```

If the student types HEART and the logical variable M is TRUE, he gets the message "PLEASE FINISH, ETC." and the logical variable HENRY is set to FALSE.

Names for logical variables are composed of letters and numerals according to the rules used for TEXT and CONTEXT names. Programmers are encouraged to use short mnemonic names (H, rather than V15 or GEORGE, for "HEART EXAMINATION", for example).

Sometimes the programmer wants to express a logically complicated relationship between several inputs. For example, he may want to know if the student has typed A AND B but NOT C, or whether the logical variables A and B are TRUE and C is FALSE. The modifiers AND, OR, and NOT may be applied to several variables to build complex expressions for use in subconditionals. For example,

```
→[I1 OR I2 OR I3→ ... ;
  ELSE  ... .].

→[NOT M→ ... ;
  A AND B→ ... ;
  ELSE→ ... .].

→[F→ ... ;
  A OR B→ ... ;
  G AND H AND J→ ... ;
  ELSE→ ... .].
```

Parentheses are not permitted for grouping, since they are used for abbreviations and for sequence conditions (see below); therefore, the modifiers cannot be mixed in a given phrase, i.e.,

NOT M OR B AND C is an illegal phrase. In each of the above examples the computer starts down the list of subconditional phrases, checking each one, until it finds one which is TRUE. When that occurs, the right side of the subconditional statement is evaluated.

The Scope of Logical Variables

Recall the expected input "YES" which occurred in two contexts in the QUIZ example. Through the CONTEXT declaration the two "YES" inputs, which have two different contextual meanings, were unambiguously separated. The reason is that the computer is "in" only one context at a time and at that time, only the inputs within that context have "meaning." Similar conditions hold for logical variables. Here are the conditions:

(1) variables defined in LET, DEFINE, and RECORD statements may be referred to <u>anywhere</u> within a TEXT, i.e., subconditionals in any context may properly contain them.

(2) variables defined with parentheses after an input have meaning (i.e., a value TRUE or FALSE) only in the context in which they are defined. Outside of that context they must be referred to in a special form.

The difference in scope of these two variable types creates a problem when names conflict. Variables created with DEFINE, LET, and RECORD statements cannot have the same spelling as variables created in parentheses. Similarly, two variables created within parentheses with the same spelling cannot exist side by side in the same context. They may, however, be created in separate contexts without conflict of meaning.

Suppose the parenthesized variable M was defined in CONTEXT Q2, and the programmer would like to use the value of this variable, M, in CONTEXT Q3. The following subconditional would have no meaning in CONTEXT Q3, since M has no meaning in Q3.

```
→[M→ ... ;
  ELSE→ ... .].
```

A special form may be used, however, to communicate values between contexts. Obviously, the form must reference the variable and the context in which it is created. In the example just given, the correct use of M in context Q3 is written,

```
→[M IN Q2→ ... ;
  ELSE→ ... .].
```

The general form of the phrase is

variablename IN contextname

This form of variable reference permits variables to be tested and used within any context without ambiguity.

Changing Context

So far the only command we have discussed which permits the machine to go from one context to another is START. When START is executed, the machine simply enters the specified CONTEXT.

Another command, called DO, is similar to START in that it causes a change in context, but it has a very different effect. When DO is executed, the machine enters the specified CONTEXT but, if it encounters a command called RETURN, the machine returns to the original context. Specifically, it returns to the action immediately following the DO. (See Fig. 2 for example.)

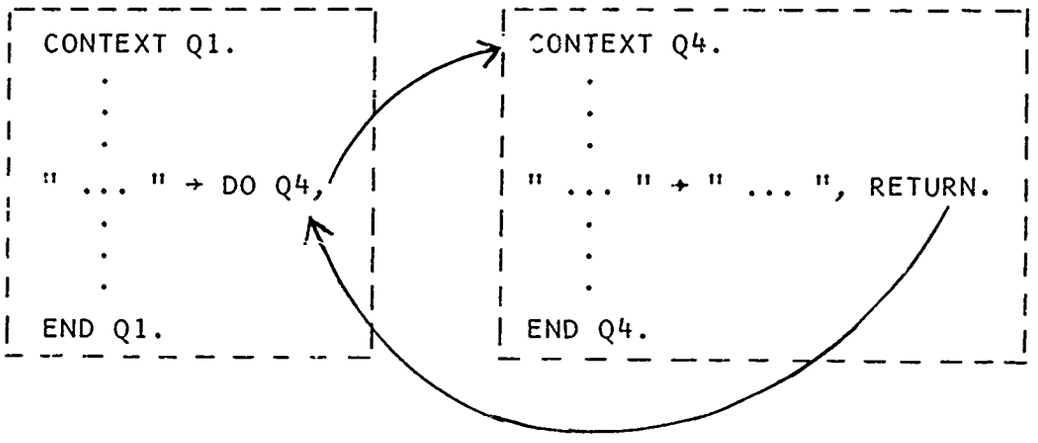


Figure 2.

RETURN occurs on the right side of a statement as an action.

The DO, RETURN commands permit the programmer to effect a temporary change of context, whose duration can be conditional. For example, in the course of the overall investigation in the mystery problem (the entire script is reproduced later) the student may have one or more conversations with each of three suspects - the wife, brother, and partner of the victim. This is accomplished by creating four contexts - INVESTIGATION, W, B, and P. Within the context INVESTIGATION the inputs WIFE, BROTHER, or PARTNER can cause (if other conditions have been satisfied) a temporary change of context to W, B, or P, respectively. In each case, a RETURN to the context INVESTIGATION occurs when the student terminates his interrogation of the suspect.

Conditionsets

Often the programmer has a long list of statements in a subconditional which he uses intact in many places within a program. To avoid writing it many times, it may be written once and given a name. It may then be referred to by its name. For example, the mystery problem contains a conditionset called ADVICE. Each element of the conditionset is a conditional statement with only variables and modifiers (AND, OR, NOT) on the left side, just as in a subconditional. When the machine enters a conditionset, it checks each variable in turn until a TRUE one is found. At that point, the right side is evaluated. There must be at least one TRUE variable in the list, hence, ELSE is always included and is always last.

A conditionset begins with the declaration CONDITIONSET followed by the name. It ends with END followed by the name. For example,

```
CONDITIONSET TEST.
VARIABLE1→ ...    .
VARIABLE2→ ...    .
  .
  .
  .
ELSE→ ...         .
END TEST.
```

A conditionset is referred to by name in CONTEXTS. The verb CHECK precedes the name. For example, from the mystery problem:

```
"LAB REPORTS" → [ALL PR→  "ON WHAT?";
                 ELSE→  CHECK ADVICE.].
```

When the student types LAB REPORTS, the machine types ON WHAT? if ALL PR is TRUE. If not, the machine enters the conditionset

ADVICE and begins checking the variables in the list until it finds the first TRUE one. Unless a statement in ADVICE instructs the machine to enter another CONTEXT, it will return to the CONTEXT in which the CHECK occurred immediately after it has checked the variable list in the conditionset and carried out the appropriate actions. Conditionsets can be entered by any CONTEXT, therefore, the variables used in it refer to the context from which the conditionset was called.

Usually, as with a CONTEXT, the checking of the statements terminates when the first TRUE statement is found. However, it is sometimes true that one wants to describe a set of mutually independent conditional statements and to carry out the actions associated with all of the TRUE ones, not just the first TRUE one. This can be accomplished by referring to a conditionset with the verb CHECK ALL.

A special action command called NULL is available for use when the programmer wants the computer to take *no action*, i.e., to do nothing. In practice, NULL is often used after the ELSE at the end of a conditionset. For example,

```
CONDITIONSET X.
M→ ...   ;
Q→ ...   ;
ELSE  NULL.
END X.
```

Sequences

In any context or conditionset, the right side of conditional statements may be expanded to direct a *sequence* of actions in numerical order. For example, in the mystery problem:

```
"MEDICAL REPORT" (M)→ 1)  "VICTIM IN COMA NEAR DEATH.";
                      2)  "NO CHANGE.".
```

When MEDICAL REPORT is typed the first time, the machine types back VICTIM, etc. The second time MEDICAL REPORT is typed (and thereafter), the machine types NO CHANGE. Each element of the sequence may be a subconditional statement and in turn each element of the subconditional may be a sequence. For example,

```
"GLASS" (GL)→ [W1 IN W→ 1)  ...   ;
                        2)  ...   .
               ALL PR→  1)  ...   ;
                        2)  ...   .
               INAPPROPRIATE→  ...   .].

"ANYTHING" (A)→ 1)  ...   ;
                2)  ...   ;
                3) [M→  ...   ;
                    P→  ...   ;
                    ELSE→  ...   .].
```

A sequence is, in effect, a special kind of logical variable whose value is set and tested by the action of the program. In principle, any degree of nesting is permitted in interleaving sequences and subconditionals. In practice, depths further than two are rarely employed by programmers.

Abbreviations

In order to minimize a great deal of typing which arises when the same output message occurs in more than one statement, the programmer can assign an abbreviation to any message. Abbreviations are enclosed in parentheses and are placed immediately after the message to be abbreviated. Once established, the

abbreviation may be used in any CONTEXT in the program. CONTEXT and CONDITIONSET names may also be abbreviated as well as messages. Each abbreviation must have unique spelling to avoid conflicting references. There is an example of abbreviations and their use in the mystery problem.

```
CONTEXT INVESTIGATION (I).
"RIFLE" (R)→ [P1→ 1)  "PRINTS BELONG TO PARTNER.", RECORD RX;
                 2)  "NO OTHER PRINTS FOUND." (FP).].

"BOTTLE" (BT)→ [W1→ 1)  "PRINTS BELONG TO WIFE.", RECORD BX;
                   2)  FP.].
END I.
```

Here INVESTIGATION is abbreviated by I in the CONTEXT declaration; NO OTHER PRINTS FOUND by FP, in the first statement. When BOTTLE is typed the first time, the machine responds PRINTS BELONG TO WIFE. The second time it is typed the machine responds, directed by the abbreviation, NO OTHER PRINTS FOUND. The abbreviation in the END declaration properly demarcates the end of the CONTEXT INVESTIGATION.

Notice that abbreviations are put in parentheses when they are created, and look exactly like logical variables. Variables and abbreviations should not be confused, however. Abbreviations stand for the literal message and have no truth value. Also, parenthesized variables are placed only after inputs whereas parenthesized abbreviations are placed only after CONTEXT names, CONDITIONSET names, and output messages.

An Example of a Mentor Script

The following Mystery Text was expressly designed as a demonstration problem to illustrate how a short but complex interaction can be written in Mentor. An associated typescript obtained from an execution of the program is reproduced in Appendix 1.2. The list of inputs available to the student is shown first. The vocabulary is deliberately small to show how few inputs are required to generate a relatively rich interaction. Many of the inputs have multiple meanings, not only because there are several contexts in which they have distinct and different relevance, but also because within a single context they elicit new responses at different times as the dialogue proceeds.

Mystery Problem

Instructions and Vocabulary

Instructions

You are an inspector from Scotland Yard. Using the Teletypewriter, you talk with suspects, with your Chief, and with investigators who can give you laboratory reports. What they say prints out on the Teletypewriter. To communicate with them, you may type any of the words and phrases listed below. Start by typing any item.

Vocabulary Items	Meanings
CONSTABLE'S REPORT	Give me the Constable's report.
MEDICAL REPORT	Give me the medical report.
SUSPECTS	Who are the suspects?
WIFE BROTHER PARTNER VICTIM SMITH	(These are the suspects whom you can interrogate.)
INTERROGATION	I wish to interrogate suspects.
WHERE WERE YOU? ALIBI? WHOM DO YOU SUSPECT? ANY EVIDENCE? THE TRUTH PLEASE THAT'S ALL FOR NOW THANK YOU	(These are questions and statements you can use for interrogating suspects.)
LAB REPORTS	I wish to receive laboratory reports.
GLASS PIPE WRENCH RIFLE BOTTLE	(These are items on which lab reports can be obtained.)
TELEPHONE?	Is there a telephone call for me?
ANY MESSAGES?	Are there any messages for me?

Mystery Script

```
Text Mystery.

Start Investigation.

Context Investigation (I).
Begin → "This afternoon Smith was found unconscious at his estate
under very suspicious circumstances.  Foul play is suspected.
You, the top inspector from the Yard, have been called in.
Investigate!".

"Constable's report" (c) → "The victim was found in the second
floor study of his home, lying between a chair and a dresser.
There was blood on his head and on the dresser.  It appeared that
he had been drinking from a glass found on a table next to the
chair.  A window at the far end of the room was shattered inward.
A cartridge was lodged in the wall at the near end of the room at
almost ceiling height.  On the premises were found a recently
fired rifle, a bloody pipe wrench, and a bottle labeled 'poison'.".

"Medical report" (m) → 1)  "Victim in coma near death.  Suffering
from wounds caused by blow to the head.  Rash in mouth indicates
possible poisoning.  Bits of shattered glass found in clothing.";
    2)  "No change.".

"Suspects" (s) → 1)  "The suspects are the victim's wife, brother,
and partner in business.  No one else was on the estate.";
    2)  "No new suspects have turned up.".

Let preliminaries (pr) = c, m, s.

"Lab reports" → [all pr → "On what?";
    Else → Check advice.].

"Glass" (gl) → [wl in w → 1)  "Fingerprints of wife and of victim
were found on glass.  Contents of glass not yet known.  Will call
you when we know.", Record gx;
    2)  "Contents still unknown.  Will inform you shortly.";
    All pr → 1)  "Fingerprints present.  Identity being checked.
Contents of glass being analyzed now.  Check again after your
interrogation.";
    2)  "Nothing new from the lab on the glass tests yet.".
    Inappropriate → Check advice.].
```

```
"Pipe wrench" (pw) → [bl in b → "The blood is not of a usual type.
We'll phone when we know what it is.  However, the prints are
definitely those of the brother.", Record px;
    All pr → 1)  "It's the bloodiest pipe wrench we've ever seen.
Checking for fingerprints.  Blood type unknown.  If you find
clues to help here, let us know.";
    2)  "It's a queer blood type.  Fingerprints may be those of
brother.  Suggest you interrogate him and check back here for
confirmation.";
    Inappropriate → Check advice.].

"Rifle" (r) → [pl in p → 1)  "Prints on rifle belong to partner.",
Record rx;
    2)  "No other prints found."  (fp);
    All pr → 1)  "Ballistics reports rifle fired the cartridge
you found.  We are checking the rifle for prints now.  Call back
in a while.";
    2)  "Prints could be those of brother or partner.  Try us
after you interrogate them.";
    3)  "No new information yet.";
    Inappropriate → Check advice.].

"Bottle" (bt) → [wl in w → 1)  "Prints on bottle established as
wife's.", Record bx;
    2)  fp;
    All pr → 1)  "Bottle contains arsenic.  Fingerprints being
checked now.";
    2)  "Prints could be those of wife or partner.  Check back
after you talk to them both.";
    3)  "Still working on prints.  Call back soon.";
    Inappropriate → Check advice.].

Let labs = gl, pw, r, bt.

"Interrogation" → [all labs → "Who do you want to see?";
    Else → Check advice.].

"Wife" → [all labs → do w; Else → Check advice.].

"Brother" → [all labs → do b; Else → Check advice.].

"Partner" → [all labs → do p; Else → Check advice.].

"Victim" or "Smith" → 1)  "Smith still in coma.";
    2)  "Still unconscious.".
```

"Telephone?" or "Any messages?" → [all labs → 1) "The chief is on the line now.", do converse.
2) "Here, sir. It's the chief again.", do converse.
3) "Here's the phone. Guess who?", do converse.
Else → "Your mother called again.".].

Else → "I don't understand you.".

End I.

Conditionset advice.

Not c → "Talk to the local constable first.".
Not m → "Find out what's wrong with the victim before doing anything else.".
Not s → "You don't even know who might have been responsible.".
Not all labs → 1) "First the lab reports on the possible weapons and accessories--the rifle, glass, pipe wrench, and bottle.";
2) "Finish all the lab work before proceeding.".

Else → Null.

End advice.

Context w.
Begin → 1) "Yes, inspector, I am Mrs. Smith.";
2) "Hi, there.".

"Where were you?" or "Alibi?" (w1) → 1) "At 12:45 I saw my husband take his heart medicine in a glass and then I went to the garden. About 10 minutes later I heard a shot and returned to find him lying unconscious on the floor.";
2) "I already told you what I know." (m1).

"Whom do you suspect?" (w2) → 1) "Why, nobody. However, you should know that my husband suspected his partner of embezzling funds. The partner will inherit the business if my husband dies.";
2) "Why, everybody.".

"Any evidence?" (w3) →[w2 → 1) "While I was upstairs I saw that bad partner -- some house guest -- pointing his rifle toward the house and lining up his sights.";
2) "You can't trust anybody these days.";

w1 → 1) "My husband will verify my story -- if he can.";
2) "Please trust me -- I was a girl scout once.";
Else → "What do you mean?" (m2).].

"The truth please" (w4) → [bx in I → 1) "You must have found out about the bottle of poison. I was too flustered to mention before that we've been invaded by ants. I was using the poison to get rid of them";
2) "You're quite insistent, aren't you, handsome?";
gx in I → 1) "Yes, I gave my husband the glass. But I swear I didn't poison him -- I love him.", Record w5;
2) "I never lie,";
Inappropriate → 1) "Whatever do you mean commissioner, honey?";
2) "I'm innocent.".].

"That's all for now" or "Thank you" → [w4 → 1) "Goodbye. Oh -- while we were talking there was a phone message for you.", Return;
2) "Goodbye. And regards to the chief when you talk to him.", Return;
Else → "Call on me again if I can be of further help.", Return.].

Else → 1) m2;
2) "I don't think that's any of your business." (m3).

End w.

Context b.
Begin → 1) "I'm Smith's brother.";
2) "Brother here.".

"Where were you?" or "Alibi?" (b1) → 1) "I was out in the orchard, near the garden, checking gopher traps when I heard a shot, around 1 o'clock. I ran back to the house and found my sister-in-law standing over my brother.";
2) m1.

"Whom do you suspect?" (b2) → 1) "My sister-in-law hated my brother. He suspected her of an affair with his partner and was threatening to disinherit her. She could have tried to shoot him from the garden. Perhaps she poisoned him.";
2) "As I told you -- Mrs. Smith.".

"Any evidence?" (b3) → [b2 → "No, but why don't you check the glass and the rifle?";
b1 → "Of course not.";
Else → m2.].

"The truth please" (b4) → [px in I → 1) "Oh, I didn't have a chance to tell you about the pipe wrench. It's really very simple. I found a gopher still alive in a trap and dispatched it with the wrench. You don't think I bopped my dear brother, do you, just because I'm in love with his wife?";
2) "What else?";
Inappropriate → "I told you the truth.".].

"That's all for now" or "Thank you" → [b4 → 1) "So long, and don't forget to check on the pipe wrench old boy.", Return;
2) "Good luck -- you may need it.", Return;
Else → "It's been a pleasure, chief.", Return.].

Else → m2.

End b.

Context p.
Begin → 1) "Smith's partner, here. Dreadful, what?";
2) "Partner speaking.".

"Where were you?" or "Alibi?" (p1) → 1) "I was out bird-watching. A rifle went off and I heard a sound of crashing window glass. I went to investigate the damage. When I got to Smith's room I saw his brother and wife laughing hysterically over his body.";
2) "Really, sir!".

"Whom do you suspect?" (p2) → 1) "Suspect? Well, sir, Mrs. Smith and her brother-in-law are rather lovey-dovey. But they're fine people, of course. And Smith was beastly to his wife, you know. Yes, I suppose she poisoned him.";
2) "No mystery at all. Mrs. Smith did it.".

"Any evidence?" (p3) → [p2 → 1) "She did try to wipe her prints from the glass -- I'm sure I saw that.";
2) "Check the bottle. I can't say more.";
p1 → m2, "Do you doubt me?";
Else → "Sir?".].

"The truth please" (p4) → [rx in I → "Well, if you must know, I wasn't bird watching, I was shooting birds -- hate them you see. My rifle went off quite accidentally you know. I'm a terrible shot, ask anyone.";
Inappropriate → 1) "Eh?";
2) m1.].

"That's all for now" or "Thank you" → [p4 → 1) "Good luck, inspector. I hope you get the cad that tried to do my dear pal in. Oh, Mrs. Smith told me that you had a telephone call. Ta ta.", Return;
2) "So long now.", Return;
Else → "You're a credit to the Yard, sir.", Return.].

Else → m3.

End p.

Context Converse.
Begin → 1) "This is the chief.", "Who is your prime suspect inspector?";
2) "Hello, inspector. This is your leader.", "Whom do you suspect now?";
3) "This is the chief.", "Well, give me your latest guess as to who's guilty.";
4) "Your superior here.", "Who dun it, bright boy?".

"Wife" → [w4 in w → 1) "Yes, the prints on the bottle were damaging and she lied about the glass. But the lab has just found that the contents of the glass were harmless -- no poison was present.", Record cw, Check test, Return;
2) "Don't you like women? Check back with me after you've got more evidence and done some thinking.", Check test, Return;
Else → "Then get the truth from her and report back.", Return.].

"Brother" → [b4 in b → 1) "Unlikely. I've just learned that the blood on the pipe wrench is gopher blood.", Record cb, Check test, Return;
2) "Come now, you've no grounds. Go to work. I'll call you later -- I expect some results.", Check test, Return;
Else → "Then make him talk and call me back.", Return.].

"Partner" → [p4 in p → 1) "That's what I thought too. However, we have just verified that a rifleman firing to the second floor with the required angle of incidence could not possibly have seen Smith.", Record cp, Check test, Return;
2) "Really? You must have some secret information. When you have a plausible case let me know.", Check test, Return;
Else → "Get him to confess. I'll check with you after you grill him.", Return.].

"Victim" or "Smith" → 1) "Enchanting idea. When you figure out how he did it, let me know.", Return;
2) "You'll be relieved from this case, bobby, if you don't succeed.", Return;

Else → "Answer me -- I want your best guess!".

End Converse.

Conditionset test.

cw and cb and cp →
"It has just been confirmed that victim had a heart condition. His prognosis is improving, dramatically. He may pull out of his coma by this evening. Hold on now -- he's starting to talk ...".
Else → Null.

End test.

End Mystery.

Appendix 1.2 Examples of Mentor Applications

This appendix is composed of transcripts illustrating Mentor applications in three different subjects—educational administration, business finance, and criminal investigation.

The educational administration problem was designed in collaboration with personnel from the University Council for Educational Administration, a confederation of over fifty universities which train more than 90% of the educational administrators in the country. The instructional program is straightforward. The user must carry out a thorough inquiry ordered in his own way. Many optional items of information are available to him. After asking for information in a number of relevant categories, he makes his decision. Depending on their particular skills and biases, users may take widely differing approaches. The program is very tolerant and does not require that the user justify his decision.

The business finance and production problem was designed in collaboration with students of the Harvard Business School. It provides a tighter and terser exchange with the user. It insists that he obtain some, but not all, of the relevant information. After he has made his decision, it is considered and examined. If he has not been careful to get all the pertinent information, he is shown the negative implications of his decision and given a chance to reconsider it.

We designed the mystery problem to show the ease with which a rich "Socratic" dialogue could be described in Mentor. We chose a familiar situation to make this demonstration generally accessible. Users not familiar with the particular terminology or procedures

of medicine, educational administration, business finance, electronics, etc. usually understand the mystery literature.

In all three transcripts, the user's inputs are preceded by a back-arrow (←) to distinguish these from the computer's responses.

Mentor Transcript from Educational Administration Problem

IT IS CHRISTMAS RECESS AND YOU ARE IN YOUR OFFICE READY TO CONFRONT AN IMMEDIATE AND PRESSING DECISION. THE QUESTION AT HAND CONCERNS THE FOLLOWING RECOMMENDATION THAT THE LATE SUPERINTENDENT HAD PREPARED FOR THE IMPENDING BOARD MEETING:

> IT IS RECOMMENDED THAT HOMEDALE ELEMENTARY SCHOOL BE CONTINUED AS A CENTER FOR KINDERGARTEN AND GRADES 1 THROUGH 6; AND THAT NECESSARY RENOVATIONS AND AN ADDITION BE MADE TO PROVIDE SPACE FOR APPROXIMATELY 450 PUPILS BY SEPTEMBER, 1966.

AS YOU SEEK THE INFORMATION NECESSARY TO ARRIVE AT A DECISION AND ADOPT A COURSE OF ACTION YOU HAVE ACCESS TO STAFF MEMBERS WHO ARE ESPECIALLY KNOWLEDGEABLE IN THE FOLLOWING AREAS:

> * COMMUNITY-POPULATION *
> * FINANCES *
> * PROGRAM AND STAFF *
> * SCHOOL PLANT *

PLEASE DIRECT YOUR QUESTIONS TO ANY OF THESE EXPERTS BY TYPING THE WORDS OR PHRASES WHICH APPEAR BETWEEN THE ASTERISKS.
WITH WHOM DO YOU WISH TO START?

←SCHOOL PLANT
HELLO, GLAD YOU CALLED. HOPE I CAN HELP YOU. THE ELEMENTARY SCHOOLS IN WORTHINGTON DISTRICT VARY AS TO * LOCATION AND SITE *, * PHYSICAL STRUCTURE *, * BUILDING CAPACITY *, * CLASSROOMS * AND * TRANSPORTATION SERVICES *. WHAT WOULD YOU LIKE TO KNOW? REMEMBER TO INDICATE THE INFORMATION YOU DESIRE BY USING THE WORD OR WORDS BETWEEN THE ASTERISKS.

←BUILDING CAPACITY
I CAN TELL YOU ABOUT * GROSS FLOOR AREA *, * PUPIL CAPACITY *, * NUMBER OF PUPILS BEING SERVED *, * % OF BUILDING UTILIZATION *,

AND * LAST YEAR'S ADM *. WHAT WOULD YOU LIKE TO HEAR?
PLEASE INDICATE WHETHER YOU DESIRE INFORMATION ABOUT THE DISTRICT (D), ONLY THE ELEMENTARY SCHOOL BUILDINGS (E), OR JUST THE HOMEDALE SCHOOL (H), BY USING THE APPROPRIATE LETTER D, E, OR H AFTER THE TERM OR TERMS INDICATING THE INFORMATION YOU DESIRE. FOR EXAMPLE, IF YOU WISHED TO KNOW THE NUMBER OF CLASSROOMS IN HOMEDALE YOU WOULD TYPE IN, 'NUMBER OF CLASSROOMS H', BUT IF YOU WANTED TO KNOW THE NUMBER OF CLASSROOMS IN ALL THE ELEMENTARY BUILDINGS YOU WOULD TYPE IN 'NUMBER OF CLASSROOMS E' AND SO ON.

←PHYSICAL STRUCTURE
I KNOW SOMETHING ABOUT * TYPE OF CONSTRUCTION *, * AGE *, AND * GENERAL CONDITION * OF SCHOOL BUILDINGS. I ALSO KNOW OF ANY * IMPROVEMENTS UNDERWAY *. WHAT WOULD YOU LIKE TO KNOW? REMEMBER TO USE 'D', 'E', OR 'H' TO INDICATE THE APPROPRIATE INFORMATION DESIRED.

←AGE H
ORIGINAL CONSTRUCTION WAS 1924; ADDITION MADE IN 1955.

←SITE
I CAN TELL YOU ABOUT * LOCATION OF SCHOOL(S) *, * AREA OF SITE(S) * AND * FUTURE PLANS * FOR THE LOCATION AND SITE OF SCHOOL BUILDING(S). WHAT INFORMATION WOULD YOU LIKE, IF ANY?

←AREA OF SITE H
3 ACRES. HOWEVER, THERE IS A 5 ACRE SITE ADJOINING THAT IS OWNED BY THE CITY. IT WAS ORIGINALLY INTENDED FOR USE IN BUILDING A HOSPITAL BUT PLANS FOR THIS HAVE BEEN ABANDONED. IT COULD BE ACQUIRED AT CITY'S COST AND PROPERLY GRADED FOR SCHOOL USE.

←GENERAL CONDITION H
THE EXTERIOR OF THE BUILDING IS IN GOOD CONDITION. INSIDE, THE KINDERGARTEN, HEALTH AND FOOD SERVICE FACILITIES ARE SATISFACTORY AS WELL AS 1Ø OF THE REGULAR CLASSROOMS. HOWEVER, 3 CLASSROOMS ARE UNSATISFACTORY AND THE ART ROOM, MUSIC ROOM, AND SPECIAL ACTIVITIES ROOM ARE JUST MINIMALLY ACCEPTABLE.

←BUILDING CAPACITY
DO YOU WANT INFORMATION ABOUT * GROSS FLOOR AREA *, * PUPIL CAPACITY *, * NUMBER OF PUPILS BEING SERVED *, * % OF BUILDING UTILIZATION * OR * LAST YEAR'S ADM *?

←CLASSROOMS
INFORMATION CONCERNING CLASSROOMS IS USUALLY ORGANIZED AS FOLLOWS: * NUMBER OF CLASSROOMS *, * NUMBER OF CLASSROOMS BEING USED *, * CONDITION OF CLASSROOMS * AND * % CLASSROOM UTILIZATION *. WHAT INFORMATION WOULD YOU LIKE?

←TRANSPORTATION SERVICES
WHAT WOULD YOU LIKE TO KNOW ABOUT * BUSSING POLICY *, * NUMBER OF BUSSES *, OR * NUMBER OF PUPILS BUSSED *? REMEMBER TO USE 'D', 'E', OR 'H' TO INDICATE THE APPROPRIATE INFORMATION DESIRED.

←NUMBER OF PUPILS BUSSED H
. . .
. . .

←COMMUNITY-POPULATION
AS IS THE CASE IN MANY OTHER COMMUNITIES CHANGES IN * POPULATION * HAVE HAD NOTICEABLE EFFECT ON SCHOOL *ENROLLMENT *. MOREOVER, DUE TO THE UNEVEN DEVELOPMENT OF * HOUSING * THESE CHANGES ARE NOT ENTIRELY SIMILAR FOR ALL AREAS. WHAT CAN I GIVE YOU MORE INFORMATION ABOUT?

←ENROLLMENT
I HAVE THE INFORMATION ORGANIZED ACCORDING TO * PAST ENROLLMENTS *, * PRESENT ENROLLMENT *, AND FUTURE * PROJECTIONS *. WHAT WOULD YOU LIKE?

←PRESENT ENROLLMENT E
2,156 IN GRADES 1-6.

←PROJECTIONS D

	LOW	HIGH
66-67	4,303	4,778
68-69	4,330	5,215
70-71	4,339	5,613

←PRESENT ENROLLMENT D
4,278 STUDENTS IN GRADES 1-12:
2,156 IN 1-6
714 IN 7-8
1,408 IN 9-12

←HOUSING
WOULD YOU LIKE INFORMATION ABOUT * HOUSING TRENDS *, * PROPOSED * HOUSING AND/OR HOUSING * SATURATION *?

←HOUSING TRENDS H
HOUSING UNITS IN THE HOMEDALE AREA HAVE BEEN BUILT AT AN INCREASING RATE EACH YEAR SINCE 1955. FROM 1955 TO 1960 THE UNITS INCREASED FROM 12/YR. TO 22/YR. DURING 1960-65 PERIOD THE RATE HAS GONE FROM 22 TO 27 UNITS.

←SATURATION H
9Ø MORE UNITS WILL BRING THIS AREA TO SATURATION.

←PROPOSED H
75 UNITS IN NEXT ' YEARS.

←HOUSING TRENDS D
IN THE PERIOD 1954 THROUGH 1964, A TOTAL OF 1,676 HOUSING UNITS WERE BUILT IN THE SCHOOL DISTRICT. INDICATIONS ARE THAT HOME BUILDING HAS NOT LEVELED OFF. THE NUMBER OF UNITS CONSTRUCTED INCREASED EACH YEAR TO THE HIGH POINT OF 362 UNITS IN 1958. IT DECREASED IN 196Ø TO 62 UNITS, BUT HAS AGAIN INCREASED EACH SUBSEQUENT YEAR.

←POPULATION
POPULATION ⁿ TRENDS ⁿ, ⁿ PRESENT POPULATION ⁿ TOTALS, OR
ⁿ BOUNDARY LINES ⁿ?

←PRESENT POPULATION D
AT PRESENT WE ESTIMATE THE CITY POPULATION AT 12,95Ø, THE ADJOINING AREAS THAT BELONG TO THE SCHOOL DISTRICT BRING THE TOTAL TO 16,43Ø.

←TRENDS D
THE POPULATION OF THE SCHOOL DISTRICT HAS GROWN RAPIDLY IN THE LAST 1Ø YEARS. HOWEVER, DUE TO THE CHANGING BOUNDARIES OF COLUMBUS, THE CHANGING BOUNDARIES OF THE SCHOOL DISTRICT, AND OTHER FACTORS, IT IS DIFFICULT TO PREDICT THE POPULATION GROWTH OF THE DISTRICT ACCURATELY FOR ANY LONG PERIOD OF TIME. THE CITY OF WORTHINGTON ITSELF IS AN INDICATOR OF THE GROWTH OF THE SCHOOL DISTRICT OVER THE LAST 14 YEARS. DURING THE 1Ø YEAR PERIOD 195Ø-196Ø THE CITY INCREASED IN POPULATION FROM 2,141 TO 9,239. HOWEVER, THE CITY BOUNDARIES AND THOSE OF THE SCHOOL DISTRICT ARE NOT COTERMINOUS.

. . .
. . .

←SCHOOL PLANT
DO YOU WANT INFORMATION ABOUT ⁿ LOCATION AND SITE ⁿ, ⁿ PHYSICAL STRUCTURE ⁿ, ⁿ BUILDING CAPACITY ⁿ, ⁿ CLASSROOMS ⁿ, OR
ⁿ TRANSPORTATION SERVICES ⁿ?

←LOCATION AND SITE
DO YOU WANT TO KNOW ABOUT, ⁿ LOCATION OF SCHOOL(S) ⁿ, ⁿ AREA OF SITE(S) ⁿ OR ⁿ FUTURE PLANS ⁿ?

←FUTURE PLANS D
HOMEDALE, COLONIAL HILLS, EVENING STREET, WILSON HILL, AND BROOKSIDE WILL BE CONTINUED IN USE AND 13 NEW ELEMENTARY CENTERS WILL BE BUILT.
RETAIN 2 JUNIOR HIGH SCHOOLS AND ADD 5 NEW ONES. RETAIN 1 HIGH SCHOOL AND ADD 3 NEW ONES.

←PROGRAM AND STAFF
DO YOU WANT INFORMATION ABOUT STAFF ↑ PERSONNEL DATA ↑, ↑ POLICIES ↑, ↑ CURRICULUM AREAS ↑, ↑ SUPPORTING SERVICES ↑ FOR PROGRAMS, OR OUR ↑ ORGANIZATION FOR INSTRUCTION ↑?

←PERSONNEL DATA
PERTINENT INFORMATION IN THIS CATEGORY IS ORGANIZED INTO: ↑ PROFILES ↑, ↑ SALARY SCHEDULE ↑, RESULTS OF A SURVEY OF STAFF ↑ ATTITUDES ↑ AND PRESENT PARTICIPATION IN ↑ PROGRAM EXPERIMENTATION ↑. WHAT INFORMATION WOULD YOU LIKE?

←SALARY SCHEDULE D

WORTHINGTON SALARY SCHEDULE

	BACHELOR'S	5 YEARS	MASTER'S
MINIMUM	$5,1ØØ	5,2ØØ	5,4ØØ
MAXIMUM	$8,1ØØ	8,34Ø	8,82Ø
INCREMENTS	12 AT 2ØØ	12 AT 21Ø	12 AT 23Ø
	2 AT 3ØØ	2 AT 31Ø	2 AT 33Ø

←FINANCES
IF YOU WOULD LIKE TO KNOW ANYTHING ABOUT ↑ TAXES ↑, ↑ BONDS ↑, OR ↑ CAPITAL OUTLAY ↑ I'M YOUR MAN.

←BONDS
FIRE AWAY. DO YOU WANT TO KNOW ABOUT BOND ↑ SALES ↑, BOND ↑ ELECTIONS ↑, BONDING ↑ CAPACITY ↑ OR BONDED ↑ INDEBTEDNESS ↑?

←ELECTIONS D
SINCE 1946 THERE HAVE BEEN SEVEN BOND ISSUES SUBMITTED TO THE ELECTORATE OF THE SCHOOL DISTRICT BY THE BOARD OF EDUCATION FOR CONSTRUCTION OR IMPROVEMENT OF EDUCATIONAL FACILITIES. ALL RECEIVED A FAVORABLE VOTE.

←TAXES
O.K. I'VE GOT THIS INFORMATION PIGEON-HOLED INTO TAX ↑ RATES ↑, ↑ TOTAL VALUATION PER PUPIL ↑, AND TAX LEVY ↑ VOTING RECORD ↑. FIRE AWAY.

←TOTAL VALUATION PER PUPIL D
$12,237.ØØ

←VOTING RECORD D
SINCE 1945 THERE HAVE BEEN ELEVEN TAX LEVIES FOR OPERATING EXPENSES SUBMITTED TO THE VOTERS OF THE SCHOOL DISTRICT BY THE BOARD OF EDUCATION. ALL RECEIVED A FAVORABLE VOTE; IN FACT, IN ONLY ONE CASE WAS THE PER CENT VOTING FOR THE ISSUE LESS THAN 6Ø PER CENT.

←CAPITAL OUTLAY
I CAN GIVE YOU THE ESTIMATED COSTS OF * NEW BUILDINGS *, * SITE ACQUISITION *, AND * BUILDING ADDITIONS * AS WELL AS THE * FUTURE PLANS * FOR CAPITAL OUTLAY.

←NEW BUILDINGS D
$58Ø,ØØØ FOR THE NEW ELEMENTARY SCHOOL

←BUILDING ADDITION D
$19Ø,ØØØ FOR ELEMENTARY
3ØØ,ØØØ FOR HIGH SCHOOL
5Ø,ØØØ FOR OTHER FACILITIES

. . .
. . .

Mentor Transcript from Business Finance Problem

YOU ARE A MANAGEMENT CONSULTANT CALLED IN TO ADVISE A TOY COMPANY ON A PROPOSAL TO CHANGE THEIR SCHEDULE OF PRODUCTION. IN THE PAST THE COMPANY'S PRODUCTION SCHEDULE HAS BEEN SEASONAL, REFLECTING THE HIGHLY SEASONAL PATTERN OF SALES. THE COMPANY IS CURRENTLY CONSIDERING THE ADVANTAGES AND RISKS INVOLVED IN A POSSIBLE SHIFT TO A LEVEL SCHEDULE OF PRODUCTION OVER THE YEAR. YOUR JUDGMENT IS BEING SOUGHT.

ASK FOR THE INFORMATION YOU WANT BY TYPING IN ITEMS FROM THE VOCABULARY LIST THAT YOU HAVE BEEN GIVEN. THIS LIST HAS A LARGE NUMBER OF ITEMS THAT ARE NOT RELEVANT TO YOUR DECISION, SOME THAT ARE OF POSSIBLE RELEVANCE, OTHERS THAT ARE VERY IMPORTANT. YOU WILL BE ENCOURAGED TO ASK PRIMARILY FOR INFORMATION YOU CONSIDER IMPORTANT IN DECIDING BETWEEN SEASONAL AND LEVEL PRODUCTION SCHEDULES.

BEGIN YOUR INVESTIGATION AS SOON AS YOU ARE READY.

←MACHINE MAINTENANCE COSTS?
THE ESTIMATED MACHINE MAINTENANCE COSTS ARE 23 THOUSAND DOLLARS FOR SEASONAL PRODUCTION AND 1Ø THOUSAND DOLLARS FOR LEVEL PRODUCTION.

THE PRO FORMA ESTIMATES MAY BE AVAILABLE NOW. GET THEM WHEN YOU HAVE THE COMPONENTS NEEDED TO MAKE THEM UP.

←OVERTIME COSTS.
THE ESTIMATED OVERTIME COSTS ARE 6Ø THOUSAND DOLLARS FOR SEASONAL PRODUCTION, NONE FOR LEVEL.

←PRO FORMA INCOME STATEMENT.
YOU HAVEN'T GOT ALL OF THE DATA NEEDED TO MAKE UP THE PRO FORMA INCOME STATEMENT. YOU DON'T KNOW THE COST OF GOODS SOLD.

←COST OF GOODS SOLD.
THE COST OF GOODS SOLD IS 6Ø PERCENT OF LAST YEAR'S SALES.

←PRO FORMA INCOME STATEMENT.
YOU HAVEN'T GOT ALL OF THE DATA NEEDED TO MAKE UP THE PRO FORMA INCOME STATEMENT. YOU DON'T HAVE THE FEDERAL INCOME TAXES FIGURE.

←FEDERAL INCOME TAXES?
THE FEDERAL INCOME TAXES ARE 5Ø PERCENT.

←INSURANCE COSTS.
THE INSURANCE COSTS ARE 4 THOUSAND DOLLARS FOR EITHER PRODUCTION SCHEDULE.

←DEPRECIATION.
DEPRECIATION EQUALS EQUIPMENT PURCHASE.

←CURRENT INVENTORY.
THE CURRENT INVENTORY IS 31 THOUSAND DOLLARS.

←CURRENT CASH.
THE CURRENT CASH IS 86 THOUSAND DOLLARS.

←MONTHLY ENDING INVENTORY.
GOOD -- YOU SHOULD ASK FOR THIS. HAVE YOU OBTAINED ALL THE ITEMS OF RAW DATA NEEDED TO CALCULATE THE MONTHLY ENDING INVENTORIES FOR LEVEL PRODUCTION?

←YES.
YOU'RE MISTAKEN. YOU HAVEN'T SEEN MONTHLY PURCHASES YET. YOU DON'T HAVE THE PROJECTED SALES FIGURES. WHEN YOU KNOW ALL OF THE RELEVANT FACTORS THE MONTHLY ENDING INVENTORIES WILL BE GIVEN TO YOU.

←MONTHLY PURCHASES.
THE MONTHLY MATERIAL PURCHASES VARY DIRECTLY WITH PRODUCTION. NO RAW MATERIAL IS ACCUMULATED IN INVENTORY.

←PROJECTED SALES?
THE PROJECTED TOTAL SALES FIGURE IS ONE MILLION DOLLARS. WOULD YOU LIKE TO SEE THE PROJECTED MONTHLY SALES ALSO?

←NO.
WHEN YOU CHANGE YOUR MIND -- AND YOU SHOULD -- YOU WILL FIND PROJECTED MONTHLY SALES IN EXHIBIT 23. HERE ARE THE MONTHLY ENDING INVENTORY FIGURES. THEY ARE 31 THOUSAND DOLLARS FOR SEASONAL PRODUCTION. THE VALUES FOR LEVEL PRODUCTION ARE SHOWN IN EXHIBIT 24.

←PAST PRODUCT MIX.
8Ø PERCENT OF THE COMPANY'S PRODUCTS HAVE HAD A STABLE HISTORY OVER THE LAST FIVE YEARS, PRIMARILY DUE TO A SINGLE PROPRIETARY ITEM OF GREAT PAST SUCCESS AND INCREASING DEMAND. THE OTHER 2Ø PERCENT ARE SHORT-LIVED NOVELTY ITEMS.

←PRO FORMA INCOME STATEMENT.
YOU HAVEN'T GOT ALL OF THE DATA NEEDED TO MAKE UP THE PRO FORMA INCOME STATEMENT. GET THE OPERATING EXPENSES FIRST.

←OPERATING EXPENSES.
FOR THE OPERATING EXPENSES SEE EXHIBIT 29.

←AVERAGE INVENTORY.
FINE. YOU'LL FIND AVERAGE INVENTORIES WITH THE MONTHLY ENDING INVENTORY DATA. ASK FOR THE PRO FORMA INCOME STATEMENT WHENEVER YOU LIKE.

←PRO FORMA INCOME STATEMENT.
HAVE YOU OBTAINED ALL THE INFORMATION YOU THINK RELEVANT TO MAKE UP THE PRO FORMA INCOME STATEMENTS FOR BOTH SEASONAL AND LEVEL PRODUCTION SCHEDULES?

←YES.

SEE EXHIBIT 4 FOR LEVEL AND EXHIBIT 22 FOR SEASONAL. THE PRO FORMA BALANCE SHEETS ARE ALSO READY NOW. BEFORE YOU SEE THEM LET'S FIND OUT IF YOU KNOW THE RELEVANT ITEMS REQUIRED TO MAKE UP A PRO FORMA BALANCE SHEET. DO YOU THINK THAT MINIMUM CASH BALANCE, PATTERN OF PURCHASES, INVENTORY LEVELS AND RETAINED EARNINGS ARE ALL REQUIRED?

←NO.
YOU'RE MISTAKEN. THEY ARE ALL REQUIRED. BUT LET ME GIVE YOU ANOTHER CHANCE. ARE DEPRECIATION, ACCRUALS, OPERATING EXPENSES AND CAPITAL EXPENDITURES REQUIRED ALSO?

←YES.
WRONG AGAIN. OPERATING EXPENSES ARE NOT DIRECTLY RELEVANT. YOU SHOULD RESTUDY THE STRUCTURE OF THE PRO FORMA BALANCE SHEET. IN CASE YOU THINK YOU KNOW ENOUGH TO CONTINUE, EXHIBIT 11 SHOWS THE PRO FORMA BALANCE SHEET FOR SEASONAL PRODUCTION. EXHIBIT 15 SHOWS THE PRO FORMA BALANCE SHEET FOR LEVEL.

NOW THAT YOU HAVE THE PRO FORMA INFORMATION YOU PROBABLY ARE READY TO STATE YOUR DECISION EITHER FOR CONTINUING WITH SEASONAL PRODUCTION OR FOR CHANGING TO LEVEL PRODUCTION. YOU MAY DO THIS DIRECTLY WHENEVER YOU ARE READY. IF YOU WANT TO GET OTHER INFORMATION FIRST, ASK FOR IT. IN CASE YOU ARE TROUBLED BY ANY OF YOUR FINDINGS AT THIS POINT OR SUBSEQUENTLY, REGISTER YOUR CONCERN BY TYPING "HELP".

←LEVEL PRODUCTION.
NOW THAT YOU'VE HAD A CHANCE TO GET ALL OF THE INFORMATION YOU WANTED AND HAVE MADE YOUR DECISION, LET'S CONSIDER IT IN THE LIGHT OF THE RELEVANT FACTS, PRESENTED AND OTHERWISE.

UNFORTUNATELY, YOU OMITTED ASKING FOR INFORMATION THAT COULD WELL BE CRITICAL IN A DECISION OF THIS KIND. YOU ACTED HASTILY, AND HAD THE GROUNDS FOR YOUR DECISION BEEN QUESTIONED, YOU MIGHT HAVE FARED BADLY. (NEVERTHELESS, YOU'LL BE GIVEN A SECOND CHANCE. YOU MAY REVIEW YOUR DECISION AFTER SEEING AND EVALUATING THE FOLLOWING RELEVANT INFORMATION THAT YOU FAILED TO OBTAIN YOURSELF.) HERE ARE THE COST FIGURES THAT YOU NEGLECTED TO LOOK INTO.

DIRECT LABOR COSTS -- 217 THOUSAND DOLLARS FOR SEASONAL, 17Ø THOUSAND FOR LEVEL. WAREHOUSING COSTS -- UNDER A THOUSAND FOR SEASONAL, ABOUT 18Ø THOUSAND FOR LEVEL.

THESE OVERSIGHTS OF YOURS TURNED OUT TO BE QUITE SIGNIFICANT. IT APPEARS THAT IT WILL COST YOU ABOUT 6Ø THOUSAND DOLLARS

(INCLUDING MACHINE MAINTENANCE COSTS) TO SHIFT TO LEVEL AND, OF COURSE, THERE ARE OTHER RISKS INVOLVED AS WELL.

CARE TO CHANGE YOUR MIND?

←NO
I SEE YOU'RE STUBBORN. I HOPE YOUR RESISTANCE IS BASED ON REASON AS WELL AS CONVICTION.

DO YOU LIKE TAKING RISKS?

←YES.
WELL, SO DO I SOMETIMES, THOUGH I'M NOT AMONG THOSE WHO PLAY BLIND MAN'S BLUFF ON TOP OF THE HIMALAYAS. WE'LL CONSIDER YOUR PERSONAL ATTITUDE TOWARD RISK ANOTHER TIME. UNTIL THEN, GOODBYE.

Mentor Transcript from Mystery Problem

THIS AFTERNOON SMITH WAS FOUND UNCONSCIOUS AT HIS ESTATE UNDER VERY SUSPICIOUS CIRCUMSTANCES. FOUL PLAY IS SUSPECTED. YOU, THE TOP INSPECTOR FROM THE YARD, HAVE BEEN CALLED IN. INVESTIGATE!

←INTERROGATION
TALK TO THE LOCAL CONSTABLE FIRST.

←CONSTABLE'S REPORT
THE VICTIM WAS FOUND IN THE SECOND FLOOR STUDY OF HIS HOME, LYING BETWEEN A CHAIR AND A DRESSER. THERE WAS BLOOD ON HIS HEAD AND ON THE DRESSER. IT APPEARED THAT HE HAD BEEN DRINKING FROM A GLASS FOUND ON A TABLE NEXT TO THE CHAIR. A WINDOW AT THE FAR END OF THE ROOM WAS SHATTERED INWARD. A CARTRIDGE WAS LODGED IN THE WALL AT THE NEAR END OF THE ROOM AT ALMOST CEILING HEIGHT. ON THE PREMISES WERE FOUND A RECENTLY FIRED RIFLE, A BLOODY PIPE WRENCH, AND A BOTTLE LABELED 'POISON'.

←MEDICAL REPORT
VICTIM IN COMA NEAR DEATH. SUFFERING FROM WOUNDS CAUSED BY BLOW TO THE HEAD. RASH IN MOUTH INDICATES POSSIBLE POISONING. BITS OF SHATTERED GLASS FOUND IN CLOTHING.

←INTERROGATION
YOU DON'T EVEN KNOW WHO MIGHT HAVE BEEN RESPONSIBLE.

←SUSPECTS
THE SUSPECTS ARE THE VICTIM'S WIFE, BROTHER, AND PARTNER IN BUSINESS. NO ONE ELSE WAS ON THE ESTATE.

←ANY MESSAGES?
YOUR MOTHER CALLED AGAIN.

←INTERROGATION
FIRST THE LAB REPORTS ON THE POSSIBLE WEAPONS AND ACCESSORIES -- THE RIFLE, GLASS, PIPE WRENCH, AND BOTTLE.

←LAB REPORTS
ON WHAT?

←GLASS
FINGERPRINTS PRESENT. IDENTITY BEING CHECKED. CONTENTS OF GLASS BEING ANALYZED NOW. CHECK AGAIN AFTER YOUR INTERROGATION.

←BOTTLE
BOTTLE CONTAINS ARSENIC. FINGERPRINTS BEING CHECKED NOW.

←RIFLE
BALLISTICS REPORTS RIFLE FIRED THE CARTRIDGE YOU FOUND. WE ARE CHECKING THE RIFLE FOR PRINTS NOW. CALL BACK IN A WHILE.

←PIPE WRENCH
IT'S THE BLOODIEST PIPE WRENCH WE'VE EVER SEEN. CHECKING FOR FINGERPRINTS. BLOOD TYPE UNKNOWN. IF YOU FIND CLUES TO HELP HERE, LET US KNOW.

←INTEROGATION
DO YOU MEAN INTERROGATION?

←YES
WHO DO YOU WANT TO SEE?

←VICTIM
SMITH STILL IN COMA.

←WIFE
YES, INSPECTOR, I AM MRS. SMITH.

←WHERE WERE YOU?
AT 12:45 I SAW MY HUSBAND TAKE HIS HEART MEDICINE IN A GLASS AND THEN I WENT TO THE GARDEN. ABOUT 10 MINUTES LATER I HEARD A SHOT AND RETURNED TO FIND HIM LYING UNCONSCIOUS ON THE FLOOR.

←ANY EVIDENCE?
MY HUSBAND WILL VERIFY MY STORY -- IF HE CAN.

←WHOM DO YOU SUSPECT?
WHY, NOBODY. HOWEVER, YOU SHOULD KNOW THAT MY HUSBAND SUSPECTED HIS PARTNER OF EMBEZZLING FUNDS. THE PARTNER WILL INHERIT THE BUSINESS IF MY HUSBAND DIES.

←ANY EVIDENCE?
WHILE I WAS UPSTAIRS I SAW THAT BAD PARTNER -- SOME HOUSE GUEST! -- POINTING HIS RIFLE TOWARD THE HOUSE AND LINING UP HIS SIGHTS.

←THE TRUTH PLEASE
WHATEVER DO YOU MEAN COMMISSIONER, HONEY?

←THAT'S ALL FOR NOW
CALL ON ME AGAIN IF I CAN BE OF FURTHER HELP.

←RIFLE
PRINTS COULD BE THOSE OF BROTHER OR PARTNER. TRY US AFTER YOU INTERROGATE THEM.

←GLASS
FINGERPRINTS OF WIFE AND OF VICTIM WERE FOUND ON GLASS. CONTENTS OF GLASS NOT YET KNOWN. WILL CALL YOU WHEN WE KNOW.

←BOTTLE
PRINTS ON BOTTLE ESTABLISHED AS WIFE'S.

←WIFE
HI, THERE.

←ALIBI?
I ALREADY TOLD YOU WHAT I KNOW.

←THE TRUTH PLEASE
YOU MUST HAVE FOUND OUT ABOUT THE BOTTLE OF POISON. I WAS TOO FLUSTERED TO MENTION BEFORE THAT WE'VE BEEN INVADED BY ANTS. I WAS USING THE POISON TO GET RID OF THEM.

←THANK YOU
GOODBYE. OH -- WHILE WE WERE TALKING THERE WAS A PHONE MESSAGE FOR YOU.

←TELEPHONE?
THE CHIEF IS ON THE LINE NOW.
THIS IS THE CHIEF.
WHO IS YOUR PRIME SUSPECT INSPECTOR?

←WIFE
YES, THE PRINTS ON THE BOTTLE WERE DAMAGING AND SHE LIED ABOUT THE GLASS. BUT THE LAB HAS JUST FOUND THAT THE CONTENTS OF THE GLASS WERE HARMLESS -- NO POISON WAS PRESENT.

←PIPE WRENCH
IT'S A QUEER BLOOD TYPE. FINGERPRINTS MAY BE THOSE OF BROTHER. SUGGEST YOU INTERROGATE HIM AND CHECK BACK HERE FOR CONFIRMATION.

←RIFLE
NO NEW INFORMATION YET.

←BROTHER
I'M SMITH'S BROTHER.

←WHERE WERE YOU?
I WAS OUT IN THE ORCHARD, NEAR THE GARDEN, CHECKING GOPHER TRAPS WHEN I HEARD A SHOT, AROUND 1 O'CLOCK. I RAN BACK TO THE HOUSE AND FOUND MY SISTER-IN-LAW STANDING OVER MY BROTHER.

←ANY EVIDENCE?
OF COURSE NOT.

←WHOM DO YOU SUSPECT?
MY SISTER-IN-LAW HATED MY BROTHER. HE SUSPECTED HER OF AN AFFAIR WITH HIS PARTNER AND WAS THREATENING TO DISINHERIT HER. SHE COULD HAVE TRIED TO SHOOT HIM FROM THE GARDEN. PERHAPS SHE POISONED HIM.

←ANY EVIDENCE?
NO, BUT WHY DON'T YOU CHECK THE GLASS AND THE RIFLE?

←THE TRUTH PLEASE
I TOLD YOU THE TRUTH.

←THANK YOU
IT'S BEEN A PLEASURE, CHIEF.

←RIFLE
NO NEW INFORMATION YET.

←PIPE WRENCH
THE BLOOD IS NOT OF A USUAL TYPE. WE'LL PHONE WHEN WE KNOW WHAT IT IS. HOWEVER, THE PRINTS ARE DEFINITELY THOSE OF THE BROTHER.

←BROTHER
BROTHER HERE.

←ALIBI?
I ALREADY TOLD YOU WHAT I KNOW.

←THE TRUTH PLEASE
OH, I DIDN'T HAVE A CHANCE TO TELL YOU ABOUT THE PIPE WRENCH. IT'S REALLY VERY SIMPLE. I FOUND A GOPHER STILL ALIVE IN A TRAP AND DISPATCHED IT WITH THE WRENCH. YOU DON'T THINK I BOPPED MY DEAR BROTHER, DO YOU, JUST BECAUSE I'M IN LOVE WITH HIS WIFE?

←THAT'S ALL FOR NOW
SO LONG, AND DON'T FORGET TO CHECK ON THE PIPE WRENCH, OLD BOY.

←ANY MESSAGES?
HERE, SIR. IT'S THE CHIEF AGAIN.
HELLO, INSPECTOR. THIS IS YOUR LEADER.
WHOM DO YOU SUSPECT NOW?

←BROTHER
UNLIKELY. I'VE JUST LEARNED THAT THE BLOOD ON THE PIPE WRENCH IS GOPHER BLOOD.

←INTERROGATION
WHO DO YOU WANT TO SEE?

←PARTNER
SMITH'S PARTNER, HERE. DREADFUL, WHAT?

←ALIBI?
I WAS OUT BIRD-WATCHING. A RIFLE WENT OFF AND I HEARD A SOUND OF CRASHING WINDOW GLASS. I WENT TO INVESTIGATE THE DAMAGE. WHEN I GOT TO SMITH'S ROOM I SAW HIS BROTHER AND WIFE LAUGHING HYSTERICALLY OVER HIS BODY.

←THE TRUTH PLEASE
EH?

←ANY EVIDENCE
WHAT DO YOU MEAN?
DO YOU DOUBT ME?

←WHOM DO YOU SUSPECT?
SUSPECT? WELL, SIR, MRS. SMITH AND HER BROTHER-IN-LAW ARE RATHER LOVEY-DOVEY. BUT THEY'RE FINE PEOPLE, OF COURSE. AND SMITH WAS BEASTLY TO HIS WIFE, YOU KNOW. YES, I SUPPOSE SHE POISONED HIM.

←ANY EVIDENCE?
SHE DID TRY TO WIPE HER PRINTS FROM THE GLASS -- I'M SURE I SAW THAT.

←THANK YOU
YOU'RE A CREDIT TO THE YARD, SIR.

←RIFLE
PRINTS ON RIFLE BELONG TO PARTNER.

←PARTNER
PARTNER SPEAKING.

←ALIBI?
REALLY, SIR!

←THE TRUTH PLEASE
WELL, IF YOU MUST KNOW, I WASN'T BIRD WATCHING, I WAS SHOOTING BIRDS -- HATE THEM YOU SEE. MY RIFLE WENT OFF QUITE ACCIDENTALLY YOU KNOW. I'M A TERRIBLE SHOT, ASK ANYONE.

←THANK YOU
GOOD LUCK, INSPECTOR. I HOPE YOU GET THE CAD THAT TRIED TO DO MY DEAR PAL IN. OH, MRS. SMITH TOLD ME THAT YOU HAD A TELEPHONE CALL. TA TA.

←ANY MESSAGES?
HERE'S THE PHONE. GUESS WHO?
THIS IS THE CHIEF.
WELL, GIVE ME YOUR LATEST GUESS AS TO WHO'S GUILTY.

←PARTNER
THAT'S WHAT I THOUGHT TOO. HOWEVER, WE HAVE JUST VERIFIED THAT A RIFLEMAN FIRING TO THE SECOND FLOOR WITH THE REQUIRED ANGLE OF INCIDENCE COULD NOT POSSIBLY HAVE SEEN SMITH. IT HAS JUST BEEN CONFIRMED THAT VICTIM HAD A HEART CONDITION. HIS PROGNOSIS IS IMPROVING, DRAMATICALLY. HE MAY PULL OUT OF HIS COMA BY THIS EVENING. HOLD ON NOW -- HE'S STARTING TO TALK ...

Appendix 2. Description of the Stringcomp Programming Language

VARIABLES

In Stringcomp, variable names must consist of a letter followed by Ø to 5 letters and/or digits.

Variables may take these types of values:

numbers
strings
true or false
indeterminate
$END

There are no restrictions on changing a variable's type or an intermixing type in the same array.

STRINGS

A string can consist of Ø to 71 characters; all printing characters on the Teletype keyboard are acceptable. String literals are expressed by putting quotation marks around the desired text, as in

X= "CHARACTER STRING"

OPERATORS

+ _ * / ↑	denoting addition, subtraction, multiplication, division, and exponentiation, respectively.

CONCATENATION

The operator "." (period) combines two strings into a new string. For example, X1= "THIS IS " .X gives X1 the string value "THIS IS A CHARACTER STRING" for X as above.

THE OPERATOR "←"

←, one of the more powerful operators, of interest mainly to the advanced user. ← causes value of the argument following it to be used in the statement in place of the ← and given argument. Examples:

```
←X="+2"
←TYPE 2←X
     2←X=4
←PRINT 3←OP 4, FOR OP= "+", "*", "↑"
 7 12 81
```

"←" is useful for establishing variable names or expressions within arguments to commands at run time. Command words (TYPE, DO, etc.) and modifiers (IF, FOR, ON, AS) will not work properly in the value of an argument to "←". To establish commands and modifiers at run time, use DO (below).

PRECEDENCE

When not modified with parentheses or brackets, the order of operations is:

1. ↑
2. * /
3. + -
4. .

Adjacent ↑'s are processed right to left; other operators, left to right.

EXPRESSIONS

Strings and numbers may be mixed in the same expression. If a string appears where a number is required, it is converted to a number (if this is not possible, an error comment occurs). For example, if X is set to the string value "2.5", then X+4 will have the value of 6.5. If a number appears where a string is required, it is converted to a string in a standard format, e.g., if X=2, X.X has the value "22".

COMMANDS

DONE, STOP, GO, REPEAT, PAGE, TO, DEFINE, HALT as in TELCOMP.

SET — Will assign a numeric or string value to a variable.

TYPE — Causes Teletype output. When TYPEing values, STRINGCOMP puts each on a separate line, preceded by the variable name or expression (except that quoted text is typed alone).

```
←TYPE 2+2,Q "THIS IS A COMMENT"
      2+2= 4
        Q= THIS IS A STRING
THIS IS A COMMENT
```

PRINT — Prints only the value of its arguments. Carriage returns are printed only if specified with the argument "#".

```
←PRINT "THE ANSWER IS ", N,#, "RIGHT?"
THE ANSWER IS 1234.7
RIGHT?
```

DEMAND

The Telcomp command that waits for the user to type in a requested value or expression. Has two new features (1) Quoted text given as an argument will be typed out:

←DEMAND "TEMPERATURE", T
TEMPERATURE
    T= 52

(2) If the word "STRING" or the symbol "$" precedes an argument, whatever characters the typist enters are treated as the string value of the variable.

←DEMAND STRING ANS
    ANS= YES

If "STRING" is not used in the demand statement, the typist must put quotes around text he wishes to enter literally.

ACCEPT

Is like DEMAND except that it does not type the quoted argument and types no carriage returns unless specified with "#".

←ACCEPT #, "GIVE ME A WORD ", $W,
" AND A NUMBER ", N
GIVE ME A WORD HELLO AND A NUMBER 483

DO

In addition to PARTS and STEPS, DO will evaluate a string valued argument, treating it as though its value had just been typed in. That is, if the argument's value is an acceptable STRINGCOMP command, it is executed; if it begins with a step number, it is made part of the program. If X="Y[1]=SIN(Q)", then DO X

will set Y[1] equal to the sine of Q. The value of X may, of course, have been previously set by the program.

```
DO "2.72".X
```

and

```
DO "2.72 TYPE X, ALPHA, ALL Y"
```

and

```
DO FOO FOR FOO = N.X
```

all give the program a new step 2.72 (assuming, that in the third case, that N is 2.72).

PLOT — Plots an ! next to the left or right side of the scale if you attempt to plot a value less than -1 or greater than 1.

PLOT ... ON X — The argument after "ON" may be numeric or string-valued. If the argument is too long, it will cover up the plotted points.

COMPARISON OPERATORS

The following operators are most often used after IF's but they can be used to create Boolean expressions. A variable can be set to a Boolean expression, and Boolean expressions can be used as arguments to and values of defined functions.

<, >, <=, =<, >=, => as in TELCOMP. (Less than, greater than, less than or equal, etc.)

=, <>, >< — Can compare two numbers, two Boolean values, two strings, or a string and a number. In

the latter case, they are considered equal if the string argument can be converted to a number equal to the numerical argument.

CTNS — "A CTNS B" is true if the string value of expression A contains the string value of expression B. Thus, the string "STRING" contains "S", "ING", and "STRING", but does not contain "STRINGA" or "GS".

IS — IS is the string matching operation. A IS B has the value *true* if A is a string expression whose value "matches" *pattern* B. A pattern consists of one or more elements joined with the concatenation operator "." or with ! which means "or".

Patterns can be composed from the following three elements:

(1) Any string expression. Matches a string segment of the same value.

(2) # Matches any string of Ø or more characters.

(3) #(A) Matches any string of A characters.

An element may be named. Thus, #[X] gives X the value of the string segment matching the #, and #(A)[X] gives X the value of the string segment matching the #(A).

Examples: "A".# matches anything beginning with "A". Q.("A"!"BB") matches the value of Q followed by "A" or "BB". Note that

parenthesis (but not brackets) may be used to delimit the range of an !.
"V".≡(2)[W].≡(4) matches anything beginning with V, followed by any two characters which are assigned as the value of W, followed by any four characters.

There are other pattern elements utilizing the ≡. ≡ may be followed immediately by one or more of the following characters, in which case only the indicated category(s) of characters will be matched.

A Letters A-Z
D Digits Ø-9
B Blanks and carriage returns
O All Other characters

For example, ≡AD(2) matches two characters which may be letters or digits.

BOOLEAN OPERATORS

AND, OR, NOT, ∧, ∨ as in TELCOMP.

FUNCTIONS

The following mathematical functions are available as in TELCOMP. SIN, COS, LOG, LN, IP, FP, XP, DP, SGN, EXP, ATN, RAN, MIN, and MAX. The following functions are new.

STL(A) STring Length. Numerical value is number of characters in A.

```
←TYPE STL("MUMBLE"), STL(498.2)
   STL("MUMBLE")= 6
      STL(498.2)= 5
```

LOC(A,B) LOCate string B in string A. Has numeric value ø if B is not in A, or the number of the first character position after the first occurrence of B in A.

```
←TYPE LOC("ABCDE", "C"), LOC ("MUM",9)
   LOC("ABCDE", "C")= 3
   LOC("MUM", 9)= ø
```

EXT(A,B) EXTract the Bth character from string A.

```
←TYPE EXT("ABC", 3)
   EXT("ABC",3)= C
```

EXT(A,B) is null if A has fewer than B characters.

EXT(A,B,C) EXTract the Bth through Cth inclusive characters from A.

```
←PRINT EXT("THIS IS A STRING", 3,7)
IS  IS
```

EXTract does not object to a third argument smaller than the second or larger than the string is long. E.G.:

EXT(STR, LOC(STR, "A"), 99)

Has the value of all of STR after the first "A".

STP(A) Has the string value of STeP A of the program. An error comment results if there is no step A.

←PRINT STP(2.72)
2.72 TYPE X, ALPHA, ALL Y

NST(A) Has the value of the Next STep number greater than A in the program, or Ø if there are no more steps.

←PRINT STP(NST (2.72))
2.8 DO PART 3

NST(Ø) has the value of the first step number in the program.

FN(A) Is equal to the text of the defined function named by the value of A.

ALPH(A,B) Is true if the string values of A and B are in alphabetical order.

NUM(A) Has the value *true* if the string A can be used as a number; that is, it is a meaningful combination of sign, digits, decimal point, and "*1Ø↑". E.G.:

←TYPE A IF NUM(A) FOR A="A", 2, 1.Ø*1Ø↑3
A= 2
A= 1ØØØ

OK(STR) If the string value is:

If the string value is:	The value of OK(STR) is:
A valid numeric expression	1
A valid string-valued expression	2
A valid true or false expression	3
An expression with value $END or indeterminate	4
None of the above	Ø

An expression is not considered valid if it contains undefined variables or functions.

EVAL(STR) The string value of STR is taken as an expression to be evaluated, and the value of this expression is returned. EVAL can be used for "indirect addressing" through strings. For instance, if the value of A is the string "B", and the value of B is the string "C", then the value of EVAL(A) is "C". However, the value of A need not be a single variable name. If A has the value "2+2", then EVAL(A)=4. EVAL duplicates some of the functions of "←".

COND(A,B,CD...) Takes an indefinite number of arguments of which the first, third, etc. must be truth-valued. Value of the function is value of expression after the first *true* odd-numbered argument.
COND(A>Ø, A, A<=Ø, Ø) has the value of A if A is positive, otherwise, it has the value Ø.

SPECIAL SYMBOLS

$L, $P, $Y, $MON, $D, $H, $MIN, $PI, $E as in TELCOMP.

$DAT Date as a string of form " 9/11/1967"

$TIM Time as a string of form " 6:04 PM"

$SPC Teletype carriage position in spaces from left margin.

$T True

$F False

Appendix 3.1 Structure of the SIMON Monitor Program

SIMON is a STRCOMP (Stringcomp) program in seventeen parts encompassing executive and problem definition functions. This section contains a description of the purpose and structure of each part, and is followed by a complete listing of all parts. Table 1 gives a very brief summary.

Part	Purpose
80	dispatcher
81	execute RUN request
82	execute COMPOSE request
83	execute CHECK request
84	execute WHY request
85	execute TYPE request
86	execute ERASE request
87	execute LIST request
88	execute EXECUTE request
89	execute HELP request
90	initialize SIMON
91	support for parts 81 and 83
92	problem definer
93	alternate for initializing SIMON
94	execute SELECT request
95	execute VALUES request
96	support for 92

Table 1. Summary of parts in SIMON.

PART 80

This is the dispatcher. It accepts a command, compares it with the list of legal instructions, and either jumps to the appropriate part or, if the command was unacceptable, complains and waits for another. If the input was an altmode, the dispatcher repeats the last command.

PART 81

This part handles the RUN commands. It checks that the student has selected his variable list and, if he hasn't, objects and returns to the dispatcher. If he has selected, it jumps to the VALUES section to get values for the selected variables. If the command was RUN YOU, it chooses randomly from the checking values to set any unspecified variables and then runs the sample program (in PART 10). If the command was RUN ME, it checks to make sure that there is a student program and then runs it. If the student has left any output variables unspecified, he gets a reminder at this point. Here, and in all subsequent parts, the control is returned to the dispatcher after execution.

PART 82

This is the COMPOSE segment. If the last command was COMPOSE, it simply types GO AHEAD and returns to the dispatcher. If the last command was a STRCOMP step, it checks the validity of the part number and either rejects it or, if the part number was between 1 and 9 inclusive, defines the step.

PART 83

The CHECK segment compares the results of the student program against those of the sample program on sets of values determined at the time of the problem definition. The number and order of

the tries are also preset. As in the RUN segment, this part checks that the student program exists and gives values to both over- and underspecified variables. For each try it checks that the answers are within a predetermined acceptable percent accuracy. It informs the student of the results (win or lose) and saves the answers from each program for use in the WHY part. If the student wins and there is another problem chained to this one, this part will load that problem and reinitialize SIMON.

PART 84

The WHY segment can only be called if a CHECK has been previously run. It types out the values used in each try of the last CHECK and prints out the results of both programs in tabular form.

PART 85

The TYPE segment allows the student to type either his whole program or individual parts, steps, or functions. It will not allow any parts outside of 1 through 9 inclusive to be examined.

PART 86

The ERASE part works just like TYPE. Any or all of the steps and parts in the student program location may be deleted, but nothing outside of this range.

PART 87

The LIST segment types out the list of acceptable SIMON commands.

PART 88

This segment handles EXECUTE commands. It will execute whatever follows the command except TYPEs, PRINTs, or DELETEs.

PART 89

This is the HELP segment. It informs the student of any over- or underspecified variables in the last SELECT list. If the student has chosen all variables correctly, it can only confirm this fact.

PART 90

This part initializes SIMON. It types out the system instructions and the specific problem instructions and starts the dispatcher.

PART 91

This is a subroutine with two separate functions. One is to handle the deletion of output variables and the other is to check the existence of the necessary output variables. It is called by CHECK and RUN.

PART 92

This is the part which handles the insertion of the problem. It gives the teacher the option of using the current variable list or manufacturing a new one. If he chooses not to use the existing list, this segment asks for a list of legal input and output variables and a short description for each. Then it accepts the list of input names required for the specific problem and checks that each is on the master list of legal variables. It accepts the number of tries (for checking), a list of input values for each try, the number of correct tries which constitute a win, and the acceptable percent error. It also lets the teacher decide whether SIMON should repeat the first set of check values (which lost on the last attempt) or go on with the next. Then it demands the list of output variables as well as the

problem instructions and a sample program. If there is another problem to be loaded after this one, the necessary changes in the CHECK part are handled here.

PART 93

This is an initialization part which does not type the system instructions, but otherwise performs as does Part 90.

PART 94

This is the SELECT segment. The student must select before running anything or checking. This part first deletes all previously selected variables and then allows the student to examine the list of acceptable variable names to choose the input and output variables he thinks he will need. The program checks that each selection is on the list of acceptable names and keeps a record of the ones he chose and those which are either unnecessary or missing.

PART 95

The VALUES segment can be called by the VALUES request or the RUN segment. It accepts values for the variable names chosen in the latest SELECT, types [in brackets] the previous value each variable had (if any), and resets the variable to that value if it receives an altmode as input.

PART 96

This is a subset of the problem insertion part (92) which facilitates editing of the variable name list.

```
SIMON-MONIT
PART 80

80.0; BEGINNING OF MONITOR LOCATION
80.01 Q2=1, RETURN=80.05
80.02 CHK=1
80.03 LOSE=SF, SELECT=SF, VALUES=SF
80.04 DO "1.0; STUDENT PROGRAM LOCATION"
80.05 ACCEPT "*",*I1;          RETURN HERE!!!!
80.06 DONE IF I1 IS "PMW"!"RCR"!"PWF"
80.065 PRINT " ",I2 IF I1=""
80.07 I1=I2 IF I1=""
80.08 TO PART 88 IF I1 IS "EXECUTE".#[Q3]
80.10 TO PART 81. IF I1 IS "RUN ".("YOU"!"ME")
80.15 TO PART 82 IF I1 IS #D(1)[Q3]."."#[Q4]
80.18 TO STEP 82.55 IF I1 CTNS "COMPOSE"
80.20 TO PART 83. IF I1 CTNS "CHECK ME"
80.25 TO PART 84. IF I1 CTNS "WHY"
80.30 TO PART 85 IF I1 IS "TYPE".#[Q3]
80.35 TO PART 86. IF I1 IS "ERASE".#[I3]
80.40 TO PART 87. IF I1 CTNS "LIST" ;LIST OF LEGAL ANSWERS
80.45 TO PART 94. IF I1 CTNS "SELECT"
80.50 TO PART 95 IF I1 CTNS "VALUES"
80.55 PRINT STIM,# IF I1 CTNS "TIME"
80.57 TO PART 89. IF I1 CTNS "HELP"
80.60 PRINT "  ?",# IF NOT I1 CTNS "TIME"
80.65 TO PART RETURN
```

```
SIMON-MONIT
PART 81

81.0; RUN SEGMENT
81.02 I2=I1
81.06 TO PART 81.91 IF NOT SELECT
81.07 DO PART 95 FOR RUN=ST IF NOT VALUES
81.08 RUN=SF, VALUES=SF
81.09 PRINT #,#
81.10 TO PART 81.50 IF I1 = "RUN ME"
81.12 TO STEP 81.20 IF UND=0
81.15 →VNAM[UNN[W8]]=TRY[1+RAN(TRYNO),UNN[W8]] FOR W8=1:1:UND
81.20 DO PART 1.
81.30 TO PART RETURN
81.51 PRINT "PUT IN A PROGRAM FIRST",# IF NST(1.)>=2.
81.52 TO STEP RETURN IF NST(1.)>2.
81.55 DO PART 91.
81.60 PRINT "YOUR PROGRAM---",#
81.70 DO PART 1.
81.73 DO PART 91.5; CHECK OUTPUT VARS
81.74 TO STEP RETURN FOR LOSE=SF IF LOSE
81.90 TO STEP RETURN
81.91 PRINT #, "YOU MUST SELECT THE VARIABLE NAMES YOU WISH TO USE"
81.92 PRINT " BEFORE YOU MAY",#,"RUN A PROGRAM.",#
81.93 PRINT "USE THE 'SELECT' COMMAND.",#
81.94 TO STEP RETURN


SIMON-MONIT
PART 82

82.0; COMPOSE SEGMENT
82.05 TO STEP 80.60 IF Q3=0
82.10 PRINT #,#
82.20 Q4=Q3."."Q4
82.25 DO Q4
82.50 TO STEP RETURN
82.55 PRINT #,#
82.60 PRINT "GO AHEAD",#,#
82.65 TO STEP RETURN
```

```
SIMON-MONIT
PART 83

83.0; CHECK SEGMENT
83.02 I2=11
83.05 PRINT #,#
83.07 COUNT=0
83.08 PRINT "PUT IN A PROGRAM FIRST",# IF NST(1.)>2.
83.09 TO STEP RETURN IF NST(1.)>2.
83.095 TO STEP 83.13 IF WHYOPT=1; REPEAT LAST TRY FIRST; OPTIONAL
83.10 02=02+1
83.12 02=1 IF 02>TRY[0,0]; TRY[0,0] HOLDS THE LAST VALUE TO TRY
83.13 ST=02 IF COUNT=0
83.15 →VNAM[W8]=TRY[02,W8] FOR W8=1:1:VNAM[0]
83.16 PRINT #,"MY PROGRAM---",#
83.18 DO PART 10.
83.20 RES10[02,W8]=→ONAM[W8]FOR W8=1:1:ONAM[0]
83.22 DO PART 91.1; DELETE OUTPUT VARS
83.23 PRINT "YOUR PROGRAM----",#
83.25 DO PART 1.
83.26 DELETE →VNAM[W6] FOR W6=1:1:VNAM[0]
83.35 DO PART 91.5; CHECK OUTPUT VARIABLES
83.36 TO STEP 83.84 FOR LOSE=SF IF LOSE
83.38 GD=0
83.42 CHK=0
83.43 RES21[02,W8] =→ONAM[W8] FOR W8=1:1:ONAM[0]
83.45 GD=GD+1 IF CK1(RES21[02,W8],RES10[02,W8]) FOR W8=1:1:ONUM
83.5 OK=GD=ONUM; 'OK' IS A TRUTH VALUED VARIABLE
83.55 COUNT=COUNT+1 IF OK
83.60 TO PART 83.8 IF COUNT =TRYNO; NUMBER OF DIFFERENT TRIES TO CHECK
83.65 PRINT "YOU'RE A LOSER",# IF NOT OK
83.7 TO STEP 83.84 IF NOT OK
83.75 TO STEP 83.10
83.78 PRINT""
83.80 PRINT "YOU'RE A WINNAH",#
83.82 TO STEP 83.96 IF NXTPRB<>""
83.84 TO STEP 83.95 IF OVR[0]=0 ∨ OK(IVNAM[OVR[1]])=0
83.9 DELETE→IVNAM[OVR[W8]]FOR O9[W8]=→IVNAM[OVR[W8]]FOR W8=1:1:OVR[0]
83.95 TO STEP RETURN
83.96 DELETE ALL VALUES,PART 10,PART 11
83.97 PRINT #,"LOADING A NEW PROBLEM",#
83.975 LOAD →NXTPRB
83.98 TO PART 93
```

```
SIMON-MONIT
PART 84

84.0; 'HY SEGMENT
84.05 PRINT #,#
84.06 PRINT "'WHY' IS TO BE USED AFTER 'CHECK ME'",# IF CHK=1
84.065 PRINT "FIX YOUR PROGRAM THEN CHECK AGAIN";# IF CHK=2
84.07 TO STEP RETURN IF CHK>=1
84.08 CAP=1
84.10 PRINT "HERE ARE THE VALUES I GOT FROM YOUR PROGRAM",#
84.17 DO PART 84.3 FOR Q3=ST:1:Q2 IF Q2>=ST
84.20 DO PART 84.3 FOR Q3=ST:1:TRY[0,0] IF Q2<ST
84.22 DO PART 84.3 FOR Q3=1:1:Q2 IF Q2<ST
84.27 TO STEP RETURN
84.30 PRINT #
84.33 DO PART 84.35 FOR E8=1:1:VNAM[0]
84.34 TO STEP 84.37
84.35 DONE IF E8=UNN[E9] FOR E9=1:1:UND IF UND<>0
84.36 PRINT VNAM[E8],"=",TRY[Q3,E8],#
84.365 DONE
84.37 PRINT IVNAM[OVR[W8]],"=",O9[W8],# FOR W8=1:1:OVR[0] IF OVR[0]<>0
84.40 PRINT "MY RESULTS:",TAB 22,"YOUR RESULTS:",# IF CAP=1
84.45 PRINT # IF CAP=0
84.46 W3=ONUM
84.5PRINT ONAM[W6],"=",RES10[Q3,W6],TAB 22,RES21[Q3,W6],#FOR W6=1:1:W3
84.70 PRINT #,#
84.80 CAP=0

SIMON-MONIT
PART 85

85.0; TYPE SEGMENT
85.02 I9=I1
85.10 TO STEP 85.50 IF Q3 CTNS "ME"
85.15 TO PART 85.60 IF Q3 IS #."STEP".#[R5]
85.20 TO PART 85.70 IF Q3 IS #."PART".#[R5]
85.25 TO PART 85.91 IF Q3 IS #."FCN".#[R5]
85.30 TO STEP 80.60
85.50 PRINT #,# "
85.52 TYPE PART R5 FOR R5=1:IP(NST(R5+.999999))-R5:9
85.55 TO STEP RETURN
85.61 TO STEP 80.60 IF NOT NUM(R5)
85.63 TO STEP 80.60 IF R5>=10 ∨ R5<1
85.65 TO STEP 80.60 IF NOT R5=NST(R5-.00001)
85.66 PRINT #,# "
85.67 TYPE STEP R5
85.69 TO STEP RETURN
85.71 TO STEP 80.60 IF NOT NUM(R5)
85.73 TO STEP 80.60 IF R5>=10 ∨ R5<1
85.75 TO STEP 80.60 IF NOT IP(R5)=IP(NST(IP(R5)-.0000!))
85.80 PRINT #,# "
85.85 TYPE PART R5
85.90 TO STEP RETURN
85.91 TYPE #,#,FCN →R5
85.95 TO STEP RETURN
```

```
SIMON-MONIT
PART 86

86.0; ERASE SEGMENT
86.02 I2=I1
86.10 TO STEP 86.50 IF @3 CTNS "ME"
86.15 TO PART 86.60 IF @3 IS #."STEP".#[R5]
86.20 TO PART 86.70 IF @3 IS #."PART".#[R5]
86.25 TO PART 86.88 IF @3 IS #."FCN".#[R5]
86.30 TO STEP 80.60
86.50 PRINT #,# "
86.52 DELETE PART R5 FOR R5=1:IP(NST(R5+.999999))-R5:9
86.55 TO STEP 86.90
86.61 TO STEP 80.60 IF NOT NUM(R5)
86.63 TO STEP 80.60 IF R5>=10 ~ R5<1
86.65 TO STEP 80.60 IF NOT R5=NST(R5-.00001)
86.66 PRINT #,#
86.67 DELETE STEP R5
86.69 TO STEP 86.90
86.71 TO STEP 80.60 IF NOT NUM(R5)
86.73 TO STEP 80.60 IF R5>=10 ~ R5<1
86.75 TO STEP 80.60 IF NOT IP(R5)=IP(NST(IP(R5)-.00001))
86.80 PRINT #,#
86.85 DELETE PART R5
86.86 TO STEP 86.90
86.88 PRINT #,#
86.89 DELETE FCN →R5
86.90 PRINT "ALL GONE",#,#
86.95 TO STEP RETURN



SIMON-MONIT
PART 87

87.0; LIST SEGMENT
87.02 I2=I1
87.05 PRINT #,#
87.1 PRINT I7[W6],# FOR W6=1:1:I7[0]
87.15 TO STEP RETURN
```

```
SIMON-MONIT
PART 88

88.0; EXECUTE SEGMENT
88.01 TO STEP 89.60 IF Q3 CTNS "TYPE"!"PRINT"!"DELETE"
88.015 PRINT #
88.02 DO Q3
88.03 PRINT #
88.04 TO STEP RETURN
```

```
SIMON-MONIT
PART 89

89.0; HELP SEGMENT
89.10 PRINT #,# FOR I2=I1
89.15 PRINT "I REALLY CAN'T HELP YOU AT THIS POINT",# IF CVALS
89.20 TO STEP RETURN IF CVALS
89.25 TO STEP 89.50 IF UND=0
89.30 PRINT "YOU HAVE NOT SELECTED ALL THE VALUES WHICH YOU NEED",#
89.35 PRINT "YOU STILL NEED TO USE:",#
89.40 PRINT VNAM[UNN[W8]]," ",# FOR W8=1:1:UND
89.45 PRINT #
89.48 TO STEP RETURN IF OVR[0]=0
89.49 PRINT "ALSO",#,#
89.50 PRINT "YOU HAVE SELECTED MORE VALUES THAN YOU WILL NEED TO"
89.55 PRINT " SOLVE THIS PROBLEM",#,"YOU DO NOT NEED:",#
89.60 PRINT VLIST[OVR[E6]],# FOR E6=1:1:OVR[0]
89.61 ;THAT IS A CROCK WHICH MUST BE FIXED, MAKE A OWN LIST!!
89.65 PRINT #
89.70 TO STEP RETURN
```

```
SIMON-MONIT
PART 90

91.0; INITIALIZATION
90.04 PRINT FORM
90.05 PRINT I8[W6],# FOR W6=1:1:I8[0]
90.06 PRINT #
90.1 PRINT I9A[W6],# FOR W6=1:1:I9A[0]
90.2 DELETE PART 1 IF OK("STP(1.0)")=2
90.30 TO PART 80.
```

```
SIMON-MONIT
PART 91

91.0; DELETES VARIABLES AND CHECKS VARIABLES
91.1 →ONAM[PWF]=0 FOR PWF=1:1:ONAM[0]; SET UP DUMMY OUTPUT VARS FOR...
91.2 DELETE →ONAM[PWF] FOR PWF=1:1:ONAM[0]; DELETION HERE
91.3 DONE
91.5 TO STEP 91.8 IF OK(ONAM[W6])=0 FOR W6=1:1:ONAM[0]
91.6 DONE
91.8 ; HANDLES UNDEFINED OUTPUT VARIABLES
91.9 PRINT "I DON'T SEE A '",ONAM[W6],"' IN YOUR PROGRAM",#
91.92 CHK=2 IF I1^CTNS "K"
91.95 LOSE=ST
```

```
SIMON-MONIT
PART 92

92.0 ;THIS PART HANDLES INSERTION OF PROBLEM
92.10 DELETE ALL VALNED FOR VALNED[0]=0
92.12 VALNED[0]=0
92.14 ACCEPT "WOULD YOU LIKE TO SEE THE CURRENT NAME LIST?  ",*ANS,#
92.16 DO STEP 94.12
92.18 ACCEPT #,"WILL YOU USE THIS LIST?  ",*ANS,#
92.20 TO STEP 92.28 IF ANS CTNS "Y"
92.22 ACCEPT "HOW MANY ENTRIES IN LIST OF POSSIBLE NAMES?  ",VLIST[0]
92.24 PRINT #,"LIST NAME AND THEN COMMENT PLEASE",#,#
92.26 ACCEPT W7." ",*VLIST[W7],TAB 10,*VLTAG[W7],# FOR W7=1:1:VLIST[0]
92.28 PRINT "LIST THE INPUT NAMES REQUIRED FOR THIS PROBLEM",# FOR W4=
92.30 ACCEPT W4." ",*VNAM[W4],#
92.32 TO STEP 92.42 FOR VNUM=W4-1 IF VNAM[W4]=""
92.34 TO STEP 92.40 IF VNAM[W4]=VLIST[W6] FOR W6=1:1:VLIST[0]
92.36 PRINT VNAM[W4]," ISN'T ON THE LIST",#
92.38 TO STEP 92.30
92.40 TO STEP 92.30 FOR W4=W4+1 FOR VALNED[W6]=1
92.42 VALNED[0]=W4-1, VNAM[0]=W4-1
92.44 ACCEPT #,"HOW MANY TRIES?  ",TRIES,#
92.46 DO PART 92.50 FOR W8=1:1:TRIES
92.48 TO STEP 92.56
92.50 PRINT #,"TRY NO.  ",W8,#
92.52 ACCEPT VNAM[W9]."=",TRY[W8,W9],# FOR W9=1:1:VNUM
92.54 DONE
92.56 TRY[0,0]=TRIES
92.58 ACCEPT #,"SHOULD I RE-TRY THE VALUE WHICH LOST FIRST? ",*WHYOPT#
92.60 WHYOPT=1 IF WHYOPT CTNS "Y"
92.62 ACCEPT#,"HOW MANY TIMES SHOULD I TRY FOR CHECKING? ",TRYNO,#
92.63 ACCEPT#,"WHAT PERCENT ERROR IS ACCEPTABLE? ",PERER8,#,#
92.64 PRINT "LIST THE OUTPUT NAMES REQUIRED FOR THIS PROBLEM",#FOR W4=
92.66 ACCEPT W4." ",*ONAM[W4],#
92.68 TO STEP 92.77 FOR ONUM=W4-1 IF ONAM[W4]=""
92.70 TO STEP 92.76 IF ONAM[W4]=VLIST[W6] FOR W6=1:1:VLIST[0]
92.72 PRINT ONAM[W4]," ISN'T ON THE LIST",#
92.74 TO STEP 92.66
92.76 TO STEP 92.66 FOR W4=W4+1
92.77 ACCEPT#,"WILL THERE BE ANOTHER PROBLEM AFTER THIS ONE? ",*G1,#
92.774 NXTPRB=""
92.775ACCEPT"WHAT IS IT FILED AS (FILE-ENTRY)? ",*NXTPRB,#IF G1 CTNS"Y
92.776 DO "83.975 LOAD ".NXTPRB IF G1 CTNS "Y"
92.78 PRINT "TYPE IN YOUR INSTRUCTIONS, END EACH LINE WITH ALTMODE",#
92.80 PRINT "TYPE  DONE  WHEN FINISHED",#,#
92.82 ACCEPT "*",*I8[W5],# FOR W5=1:1 UNTIL I8[W5-1]="DONE"
92.84 I8[0]=W5-2, ONAM[0]=ONUM
92.85 DO "11; MORE SPACE FOR SAMPLE PROBGRAM"
92.855 FILEPB="ALL VALUES,PART 10,PART 11,STEP 83.975,ALL FCNS"
92.856PRINT #,"FILE THIS PROBLEM BY TYPING 'FILE →FILEPB AS ...'",#,#
92.86 PRINT "THE EXAMPLE GOES IN PART 10.",#,#,#
```

```
SIMON-MONIT
PART 93

93.0; START-UP LOCATION TO GET SIMON W/O SYSTEM INSTRUCTIONS
93.1 PRINT FORM
93.2 PRINT I8[W6],# FOR W6=1:1:I8[0]
93.25 DELETE PART 1  IF OK("STP(1.0)")=2
93.3 TO PART 80
93.97 PRINT #,"LOADING A NEW PROBLEM",#
93.975 LOAD →NXTPRB
```

```
SIMON-MONIT
PART 94

94.03 SELECT SEGMENT
94.01 TO STEP 94.03 IF OK("IVNUM")=0
94.02 DELETE →IVNAM[W6] IF OK(IVNAM[W6])<>0 FOR W6=1:1:IVNUM
94.03 DELETE UND,ALL OVR,ALL UNN FOR OVR[0]=0 FOR UND=0 FOR UNN[0]=0
94.05 PRINT #,#
94.06 DELETE ALL VALUSD, ALL IVNAM FOR VALUSD[0]=0 FOR IVNAM[0]=0
94.07 I2=I1
94.08 VALUSD[0]=0,IVNAM[0]=0,OVR[0]=0,UND=0
94.10 ACCEPT "SHALL I LIST THE LEGAL VARIABLE NAMES?  ",*ANS,#
94.12PRINT VLIST[W4],TAB 8,VLTAG[W4],#FOR W4=1:1:VLIST[0]IF ANS CTNS"Y
94.15 PRINT "HOW MANY INPUT VALUES DO YOU THINK YOU WILL NEED?  "
94.16 ACCEPT IVNUM,#
94.19 PRINT "EH?",#IF IVNUM<=0 OR FP(IVNUM)<>0
94.20 TO STEP 94.16 IF IVNUM<=0 OR FP(IVNUM)<>0
94.22 PRINT "LIST THEM",# FOR W6=1 IF IVNUM<>1
94.23 PRINT "LIST IT",# FOR W6=1 IF IVNUM=1
94.24 ACCEPT W6." ",*IVNAM[W6],#
94.26 TO STEP 94.275IF IVNAM[W6]=VLIST[W7]FOR W7=1:1:VLIST[0]
94.27 TO STEP 94.80 FOR RET=94.24
94.275 VALUSD[W7]=1
94.277 OVR[0]=OVR[0]+1,OVR[OVR[0]]=W7 IF VALNED[W7]<>1
94.28 TO STEP 94.24 FOR W6=W6+1 IF W6<IVNUM
94.285 VALUSD[0]=IVNUM, IVNAM[0]=IVNUM
94.286 DO PART 94.288 FOR E4=1:1:VNUM FOR E5=0
94.287 TO STEP 94.31
94.288 DONE IF VNAM[E4]=IVNAM[E6] FOR E6=1:1:IVNUM
94.30 UNN[E5]=E4, UND=E5 FOR E5=E5+1
94.305 DONE
94.31 PRINT #,"HOW MANY OUTPUT VARIABLE NAMES?  "
94.32 ACCEPT OVNUM,#
94.35 PRINT "EH?",#IF OVNUM<=0 OR FP(OVNUM)<>0
94.36 TO STEP 94.32 IF OVNUM<=0 OR FP(OVNUM)<>0
94.38 PRINT "LIST THEM",# FOR W6=1 IF OVNUM<>1
94.39 PRINT "LIST IT",# FOR W6=1 IF OVNUM=1
94.40 ACCEPT W6." ",*OVNAM[W6],#
94.42 TO STEP 94.46 IF OVNAM[W6]=VLIST[W7] FOR W7=1:1:VLIST[0]
94.44 TO STEP 94.80 FOR RET=94.40
94.46 TO STEP 94.40 FOR W6=W6+1 IF W6<OVNUM
94.47 CVALS=UND+OVR[0]=0
94.48 TO STEP RETURN FOR SELECT=ST
94.52 TO STEP RETURN
94.80 PRINT IVNAM[W6] IF RET=94.24
94.82 PRINT OVNAM[W6] IF RET=94.40
94.84 PRINT "  ISN'T ON THE LIST",#,"DO YOU WANT TO SEE THE LIST  "
94.86 ACCEPT *ANS,#
94.88 DO STEP 94.12
94.90 TO STEP RET
```

```
SIMON-MONIT
PART 95

95.0; VALUES SEGMENT
95.05 PRINT #,#
95.10 I2=I1
95.15 W6=1
95.20 →IVNAM[W7]="" IF OK(IVNAM[W7])=0 FOR W7=1:1:IVNUM
95.25 PRINT IVNAM[W6],"= "
95.30 PRINT "[".→IVNAM[W6]."]",TAB 11,"= " IF →IVNAM[W6]<>""
95.35 ACCEPT *F002U
95.36 PRINT →IVNAM[W6] IF F002U =""
95.40 →IVNAM[W6]=F002U IF F002U<>""
95.42 PRINT #
95.45 TO STEP 95.25 FOR W6=W6+1 IF W6<IVNUM
95.47 DONE IF RUN; RETURN TO RUN SEG IF CALLED FROM THERE
95.50 TO STEP RETURN FOR VALUES=ST


SIMON-MONIT
PART 96

96.1 ; ENTER AND UPDATE VLIST AND VLTAG
96.15 ACCEPT "WHOLE NEW LIST? ",*ANS1,#
96.16 TO PART 96.7 IF ANS1 CTNS "Y"
96.20 ACCEPT "ADD TO END OF EXISTING LIST? ",*ANS1,#
96.22 TO PART 96.6 IF ANS1 CTNS "Y"
96.24 ACCEPT "EDIT EXISTING LIST? ",*ANS1,#
96.26 DONE IF NOT ANS1 CTNS "Y"
96.28 ACCEPT "SHALL I TYPE THE LIST? ",*ANS1,#
96.3PRINT VLIST[W4],TAB 8,VLTAG[W4],#FOR W4=1:1:VLIST[0]IF ANS1 CTNS"Y
96.32 PRINT #,#
96.33 W4=1
96.34 PRINT VLIST[W4],TAB 8,VLTAG[W4],"   "
96.36 ACCEPT "OK? ",*ANS1,#
96.37 ACCEPT "NAME ",*VLIST[W4] IF NOT ANS1 IS "Y"!"YES"!""
96.38 ACCEPT "   COMMENT ",*VLTAG[W4],# IF NOT ANS1 IS "Y"!"YES"!""
96.40 TO STEP 96.2 IF W4=VLIST[0]
96.42 TO STEP  96.34  FOR W4=W4+1
96.60 ;ADD TO END OF EXISTING LIST
96.61 DO STEP W6 FOR W6=96.28, 96.30,96.76
96.63 TO STEP 96.78 FOR W4=VLIST[0]+1
96.70; WHOLE NEW LIST
96.72 DELETE ALL VLIST,ALL VLTAG FOR VLIST[0]=0 FOR VLTAG[0]=0
96.74 PRINT "TYPE VARIABLE NAME AND VERY SHORT DESCRIPTION E.G",#
96.75 PRINT #,"T0       INITIAL TIME",#,#
96.76 PRINT "TYPE 'DONE' WHEN FINISHED ENTERING LIST",#,#
96.77 W4=1
96.78 ACCEPT *VLIST[W4],TAB 8,*VLTAG[W4],#
96.80 TO STEP 96.20 FOR VLIST[0]=W4-1 IF VLIST[W4]CTNS "DONE"
96.81 ;
96.82 TO STEP 96.78 FOR W4=W4+1
```

Appendix 3.2 Information Required by SIMON for the Rescue Problem.

```
←LOAD PHIL-DROP1
P? RESCUE PROBLEM PART ?
←TYPE PART 10
10.00 DONE IF I1 IS "RUN ME";MY PROGRAM
10.01 TO STEP 10.025 IF I1 CTNS "CHECK"
10.02 ACCEPT#,"DO YOU WANT TO SEE THE TRAJECTORIES PLOTTED? ",*ANS
10.025 AXP=0,AYP=0; NO ACCELERATIONS IN X AND Y YET
10.03 TYPE "HELICOPTER TOO LOW; TROUBLE." IF ZHO+VZH*TR<=100
10.04 DONE IF ZHO+VZH+TR<=100
10.05 XPO=XHO+VXH*TR,YPO=YHO+VYH*TR,ZPO=ZHO+VZH*TR
10.06 TYPE "THE PACKAGE WILL NOT LAND." IF ZPO<=C@AZP>0
10.07 DONE IF ZPO<=0@AZP>0
10.08 BPR=VZH/AZP,C=2*ZPO/AZP,RT=-BPR+SQRT(BPR↑2-C),TL=RT+TR
10.09 XP=XPO+VXH*RT+.5*AXP*RT↑2,YP=YPO+VYH*RT+.5*AYP*RT↑2
10.095 ZP=ZPO+VZH*RT+.5*AZP*RT↑2
10.1 DIS=SQRT((100-XP)↑2+(YP)↑2)
10.11 TYPE TL,XP,YP,ZP,DIS
10.12 DO PART 11 IF ANS CTNS "Y" IF NOT I1 CTNS "CHECK"

←TYPE PART 11
11; MORE SPACE FOR SAMPLE PROGRAM
11.01 DT=(TL-TR)/10,T=TR-2*DT
11.02 MX=COND(VZH<=0,ZHO+VZH*T,ST,ZHO+VZH*(TL+DT))
11.025PRINT#,"X POSITION  0 FEET",TAB 60,IP(MX)," FEET",#
11.03;
11.04 XH=XHO+VXH*T,ZH=ZHO+VZH*T
11.05 ZP=ZH IF T<=TR
11.06 ZP=ZPO+VZH*(T-TR)+AZP*(T-TR)↑2*.5 IF T>TR
11.07 PLOT 2*ZP/MX-1,4,#,1,-1,#,2*ZH/MX-1 ON XH
11.08 TO STEP 11.03 IF T<TL+2*DT FOR T=T+DT


←TYPE VLIST[I] FOR I=1:1:VLIST[0]
     VLIST[1]= XHO
     VLIST[2]= YHO
     VLIST[3]= ZHO
     VLIST[4]= VXH
     VLIST[5]= VYH
     VLIST[6]= VZH
     VLIST[7]= TR
     VLIST[8]= AZP
     VLIST[9]= TL
    VLIST[10]= XP
    VLIST[11]= YP
    VLIST[12]= ZP
    VLIST[13]= DIS
    VLIST[14]= MP
```

```
←TYPE VNAM[I] FOR I=1:1:VNAM[0]
      VNAM[1]= XHO
      VNAM[2]= YHO
      VNAM[3]= ZHO
      VNAM[4]= VXH
      VNAM[5]= VYH
      VNAM[6]= VZH
      VNAM[7]= TR
      VNAM[8]= AZP

←TYPE ONAM[I] FOR I=1:1:ONAM[0]
      ONAM[1]= TL
      ONAM[2]= XP
      ONAM[3]= YP
      ONAM[4]= ZP
      ONAM[5]= DIS

←TYPE TRY[I,J] FOR J=1:1:VNAM[0] FOR I=1:1:TRY[0,0]
     TRY[1,1]=   -300
     TRY[1,2]=     20
     TRY[1,3]=   1000
     TRY[1,4]=     20
     TRY[1,5]=      2
     TRY[1,6]=    -50
     TRY[1,7]=     10
     TRY[1,8]=    -32

     TRY[2,1]=   -276
     TRY[2,2]=    400
     TRY[2,3]=    300
     TRY[2,4]=     20
     TRY[2,5]=     -2
     TRY[2,6]=     50
     TRY[2,7]=     10
     TRY[2,8]=    -32

     TRY[3,1]=  -1000
     TRY[3,2]=  -1000
     TRY[3,3]=   1000
     TRY[3,4]=     50
     TRY[3,5]=     50
     TRY[3,6]=    -50
     TRY[3,7]=     10
     TRY[3,8]=    -32

     TRY[4,1]=   -100
     TRY[4,2]=   -100
     TRY[4,3]=    500
     TRY[4,4]=      5
     TRY[4,5]=      5
     TRY[4,6]=    -25
     TRY[4,7]=      8
     TRY[4,8]=    -32
```

```
←LOAD PHIL-DROP2
PF RESCUE PR\BLEM PART 2 (VALUES ONLY)
←TYPE PART 10
10.00 DONE IF I1 IS "RUN ME";MY PROGRAM
10.01 TO STEP 10.03 IF I1 CTNS 'CHECK"
10.02 ACCEPT#,"DO YOU WANT TO SEE THE TRAJECTORIES PLOTTED? ",*ANS
10.03 TYPE "HELICOPTER TOO LOW; TROUBLE." IF ZHO+VZH*TR<=100
10.04 DONE IF ZHO+VZH+TR<=100
10.05 XPO=XHO+VXH*TR,YPO=YHO+VYH*TR,ZPO=ZHO+VZH*TR
10.06 TYPE "THE PACKAGE WILL NOT LAND." IF ZPO<=0@AZP>0
10.07 DONE IF ZPO<=0@AZP>0
10.08 BPR=VZH/AZP,C=2*ZPO/AZP,RT=-BPR+SQRT(BPR↑2-C),TL=RT+TR
10.09 XP=XPO+VXH*RT+.5*AXP*RT↑2,YP=YPO+VYH*RT+.5*AYP*RT↑2
10.095 ZP=ZPO+VZH*RT+.5*AZP*RT↑2
10.1 DIS=SQRT((100-XP)↑2+(YP)↑2)
10.11 TYPE TL,XP,YP,ZP,DIS
10.12 DO PART 11 IF ANS CTNS "Y" IF NOT I1 CTNS "CHECK"

←TYPE PART 11
11.01 DT=(TL-TR)/10,T=TR-2*DT
11.02 MX=COND(VZH<=0,ZHO+VZH*T,ST,ZHO+VZH*(TL+DT))
11.025PRINT#,"X POSITION  0 FEET",TAB 60,IP(MX)," FEET",#
11.03 TP=(XPO+VXH*(T-TR)+.5*AXP*(T-TR)↑2-XHO)/VXH IF T>TR
11.031 TP=T IF T<=TR
11.04 XH=XHO+VXH*TP,ZH=ZHO+VZH*TP
11.05 ZP=ZH IF T<=TR
11.06 ZP=ZPO+VZH*(T-TR)+AZP*(T-TR)↑2*.5 IF T>TR
11.07 PLOT 2*ZP/MX-1,#,#,1,-1,#,2*ZH/MX-1 ON XH
11.08 TO STEP 11.03 IF T<TL+2*DT FOR T=T+DT


←TYPE VLIST[I] FOR I=1:1:VLIST[0]
    VLIST[1]= XHO
    VLIST[2]= YHO
    VLIST[3]= ZHO
    VLIST[4]= VXH
    VLIST[5]= VYH
    VLIST[6]= VZH
    VLIST[7]= MP
    VLIST[8]= TR
    VLIST[9]= AXP
   VLIST[10]= AYP
   VLIST[11]= AZP
   VLIST[12]= TL
   VLIST[13]= XP
   VLIST[14]= YP
   VLIST[15]= ZP
   VLIST[16]= DIS
```

```
←TYPE VNAM[I] FOR I=1:1:VNAM[0]
      VNAM[1]= XHO
      VNAM[2]= YHO
      VNAM[3]= ZHO
      VNAM[4]= VXH
      VNAM[5]= VYH
      VNAM[6]= VZH
      VNAM[7]= TR
      VNAM[8]= AXP
      VNAM[9]= AYP
     VNAM[10]= AZP

←TYPE ONAM[I] FOR I=1:1:ONAM[0]
      ONAM[1]= TL
      ONAM[2]= XP
      ONAM[3]= YP
      ONAM[4]= ZP
      ONAM[5]= DIS
```

```
←TYPE TRY[I,J] FOR J=1:1:VNAM[0] FOR I=1:1:TRY[0,0]
    TRY[1,1]=   -200
    TRY[1,2]=   -200
    TRY[1,3]=    500
    TRY[1,4]=     10
    TRY[1,5]=     10
    TRY[1,6]=    -20
    TRY[1,7]=     10
    TRY[1,8]=      2
    TRY[1,9]=      3
   TRY[1,10]=    -32

    TRY[2,1]=   -500
    TRY[2,2]=   -500
    TRY[2,3]=    500
    TRY[2,4]=     20
    TRY[2,5]=     20
    TRY[2,6]=    -20
    TRY[2,7]=     10
    TRY[2,8]=     -2
    TRY[2,9]=     -3
   TRY[2,10]=    -32

    TRY[3,1]=   -200
    TRY[3,2]=   -300
    TRY[3,3]=   5000
    TRY[3,4]=     10
    TRY[3,5]=     15
    TRY[3,6]=    -32
    TRY[3,7]=     10
    TRY[3,8]=      2
    TRY[3,9]=      3
   TRY[3,10]=    -32

    TRY[4,1]=      0
    TRY[4,2]=      0
    TRY[4,3]=    500
    TRY[4,4]=      2
    TRY[4,5]=      2
    TRY[4,6]=    -20
    TRY[4,7]=      1
    TRY[4,8]=      2
    TRY[4,9]=      2
   TRY[4,10]=    -32
```

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author)	2a. REPORT SECURITY CLASSIFICATION
Bolt Beranek and Newman Inc. 50 Moulton Street Cambridge, Mass. 02138	UNCLASSIFIED 2b. GROUP

3. REPORT TITLE

COMPUTER SYSTEMS FOR TEACHING COMPLEX CONCEPTS

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)

Final Report 1 October 1963 - 30 September 1968

5. AUTHOR(S) (First name, middle initial, last name)

Wallace Feurzeig

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
March 1969	183	

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
Nonr-4340(00) b. PROJECT NO.	BBN Report No. 1742
c.	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)
d.	

10. DISTRIBUTION STATEMENT

This document has been approved for public release and sale; its distribution is unlimited. Reproduction in whole or in part is permitted for any purposes of the United States Government.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	Department of the Navy Office of Naval Research Washington, D. C. 20360

13. ABSTRACT

This research concerns various ways of using computers for teaching problem-solving concepts and skills. New lines of approach to programmed teaching, programming, and instructional monitoring were investigated in various instructional contexts including mathematics, physics, and medicine.

Four programming systems--Mentor, Stringcomp, Simon, and Logo--were designed and used as an integral part of these investigations. The systems are described and their capabilities demonstrated in instructional applications of several kinds.

The work suggests some new ways in which computers might make valuable contributions to education. (1) The teaching of appropriate programming languages can provide a conceptual and operational framework for the teaching of mathematics. (2) Utilizing diagnostic cues, instructional monitors can enhance the teaching of practical subjects (navigation, languages, music) whose mastery requires the integration of mechanical and intellectual skills.

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Computers and education						
Computer-assisted instruction						
Programmed instruction						
Programmed teaching						
Programmed testing						
Programming languages						
Instructional monitors						
Learning						
Mathematics education						
Medical education						
Simulation						
Problem-solving						